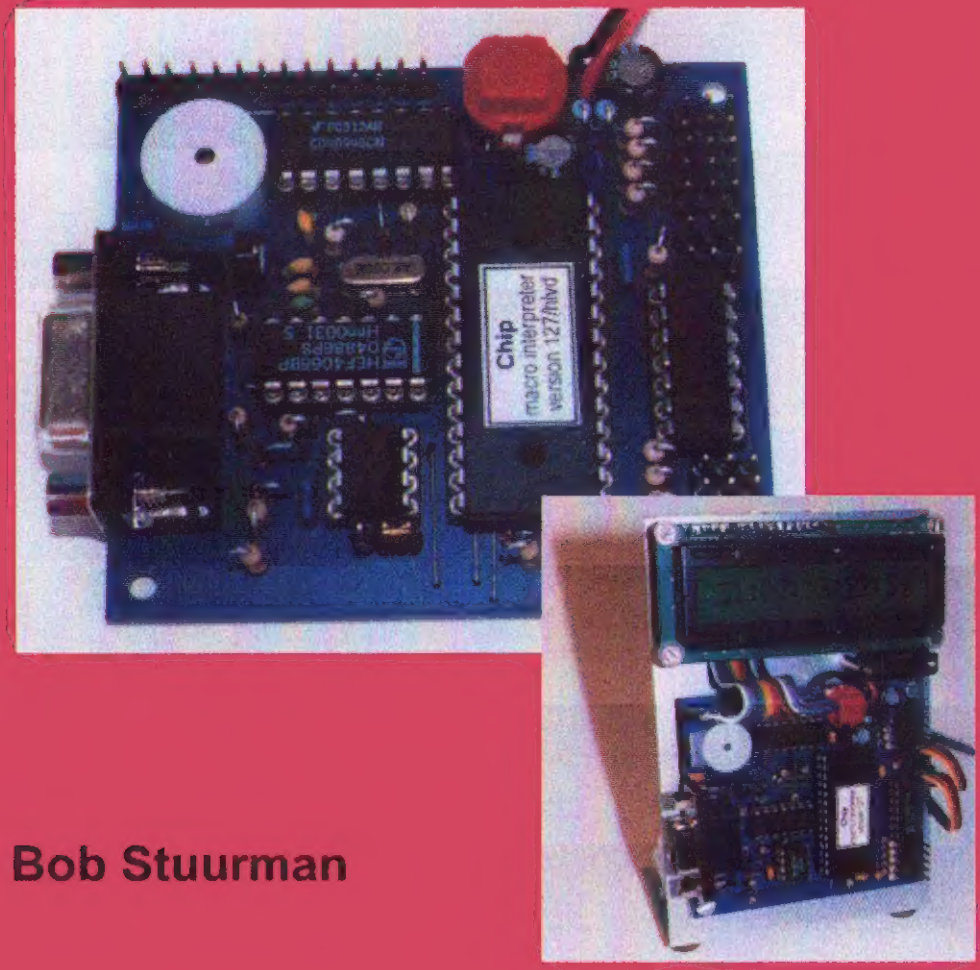
Chip

een zelfbouw computertje

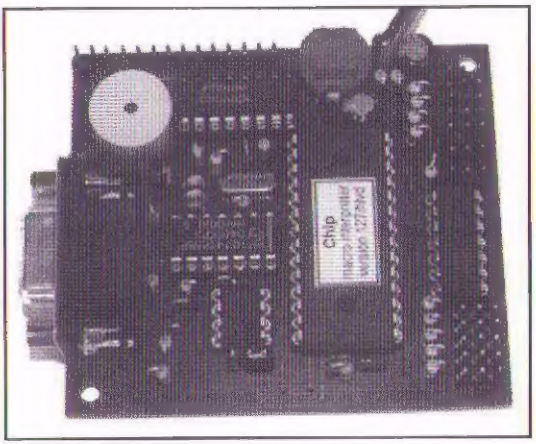


Bob Stuurman

Vego

Chip

een zelfbouw computertje



Bob Stuurman

Vego VOF

Postbus 32.014, 6370 JA Landgraaf (NL)
vego_vof@compuserve.com, www.vego.nl

Auteur
Bob Stuurman Weesp (NL)

Uitgever
Vego VOF Postbus 32.014, 6370 JA Landgraaf (NL)
E-mail vego_vof@compuserve.com
Telefoon 045-533.22.00
Fax 045-533.22.02

Elektronische pagina-opmaak
Vego VOF, Landgraaf http://www.vego.nl

POD-productie
CPF Landgraaf http://www.cpflandgraaf.nl

Eerste druk
April 2004

ISBN
90-805610-9-6

NUR
958

SISO
523.1

DISCLAIMER
Auteur en uitgever zijn zich volledig bewust van hun taak een zo betrouwbaar mogelijke uitgave te verzorgen. Voor eventueel in deze uitgave voorkomende onjuistheden kunnen zij echter geen aansprakelijkheid aanvaarden.

Inhoud

Inleiding

Chip is een computertje ongeveer zo groot als twee luciferdoosjes tegen elkaar. Het hart van Chip is een microcontroller. Heel bijzonder is, dat in de microcontroller geen programma zit voor een vaste taak, maar een programma dat instructies in een hogere programmeertaal decodeert en ze in microcontroller code uitvoert.
Deze programmeertaal is speciaal geschreven voor Chip en heel eenvoudig van opzet.
Om met Chip te werken is kennis van microcontrollers niet nodig. Het enige dat nodig is, is Chip's instructieset.

Chip bevat standaard alle hardware die voor eenvoudige toepassingen nodig is en voor de sturing daarvan bevat de hogere programmeertaal instructies.

Om Chip te gebruiken is alleen een PC nodig met een seriële poort. Chip bevat alle software die voor de communicatie nodig is. Het programmeren van Chip kan direct in de hogere programmeertaal, aan de hand van de instructieset, maar ook in de symbolische schrijfwijze met **mnemonics** en **labels**. Door een **assembler** worden de symbolische programma's in instructies vertaald en kunnen dan meteen in Chip worden geladen.
Deze manier van programmeren levert heel leesbare programma's en maakt de kans op fouten kleiner.

Ondanks alle eenvoud is Chip een vrij krachtig computersysteem. Maar om alle mogelijkheden vol te benutten is enige kennis van elementaire digitale principes onmisbaar. Ook begrippen als bytes, nibbles en bits moeten vertrouwd klinken en de elementaire werking van het computermodel volgens von Neumann moet bekend zijn.

Als aan deze voorwaarden wordt voldaan, zal men van Chip veel plezier kunnen hebben en het kleine apparaatje steeds meer gaan waarderen.

Bob Stuurman
april 2004

1 Hardware en operating system

Inleiding

Chip is een computertje dat speciaal is ontworpen om kleine projecten te automatiseren. Dat kan een woning zijn, een alarminstallatie, een robot, een datalogger, een weerstation, kortom systemen waarin de tijd een rol kan spelen en waar signalen moeten worden gemeten en relais en/of servo's aangestuurd. Het hart van Chip is een microcontroller, die een operating system bevat voor de communicatie met de gebruiker én een interpreter voor het uitvoeren van programma's, die in een interne EEPROM zijn opgeslagen. Zo'n programma bestaat uit een reeks macro-instructie's, elk twee bytes lang, die de in machinetaal geschreven routines in de microcontroller oproepen en sturen. Een Chip programma kan gewoon in "mensentaal" worden geschreven. Een assembler stelt de macro's samen, die dan met een uploader in de EEPROM worden geschreven. Om met Chip te kunnen werken is een PC nodig met een seriele poort, dat is alles.
Een aardige toepassing is die van een robot, en dat zullen we in een volgend hoofdstuk verder uitwerken. Chip is ook heel geschikt om woningen te automatiseren. Zo kan de verwarming worden geregeld aan de hand van de binnen- en buitentemperaturen, de dag van de week en de tijd. Op een display worden de gegevens zichtbaar gemaakt en met een toetsenbordje kunnen bedieningsfuncties worden opgeroepen. Als u de Basic Stamp [1] kent, dan is het u misschien opgevallen dat tot zover Chip wel enige gelijkenis vertoont met de Basic Stamp zélf. Toch is dat zuiver toeval, want het concept van de in Chip gebruikte macrotaal is niet nieuw[2].

Specificaties

De voornaamste eigenschappen van Chip kunnen als volgt worden samengevat:

- kenmerk:
  programmeerbare minicomputer op basis van een ST62T65 microcontroller
- programmeertaal:
  macro-instructies van 2 bytes
- type vertaler:
  interpreter

- aantal variabelen:
  16, 0 tot en met F
- instructietijd:
  ongeveer 1 ms, afhankelijk van type instructie
- programmageheugen:
  I^2C EEPROM met 2.048 bytes
- ingangen:
  vijf, digitaal en analoog leesbaar
- keyboard:
  twaalf key keyboard mogelijk op analoge ingang
- uitgangen:
  - vijf, digitaal met 20 mA sink/7 mA source (begrensd door weerstanden)
  - twee servo's, pulstijd 0,9 ms tot en met 1,9 ms, frequentie 50 Hz
  - alternatief voor servo-uitgangen is één analoge uitgang
  - LCD-display, 1 regel 16 karakters
  - sounder
- lopende tekst:
  maximaal 24 karakters op het LCD
- real-time klok:
  weken, dagen, uren, minuten en seconden
- klok fine-tuning:
  softwarematig in stappen van 2,7 ppm
- timers:
  korte timer, seconden en minuten timers en een sounder
- communicatie:
  interactieve commandoprocessor, RS232, protocol 19.200,7,n,2
- voedingsspanning:
  4,4 V tot 6,0 V
- voedingsbron:
  accupakket van 4 NiCad's of 5 V voeding
- stroomverbruik:
  ongeveer 3 mA zonder externe componenten
- print:
  enkelzijdig, 70 x 62 mm^2
- hulp software:
  Chip assembler, Chip terminal met hex uploader
- toepassingen:
  woning automatisering, robots, lange tijd logger, eenvoudige besturingen, alarminstallaties, technische modelbouw

Systeembeschrijving

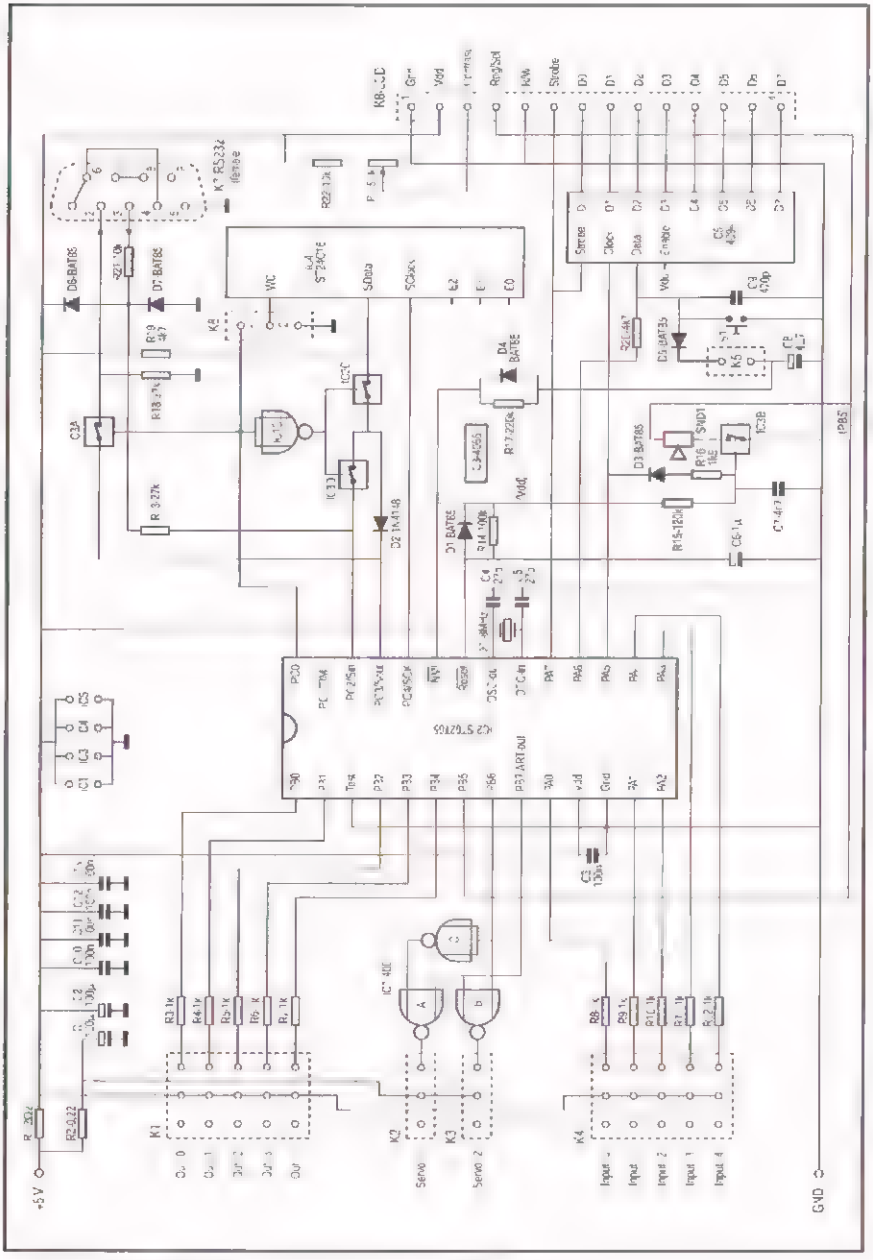
Chip bevat niet alleen een geprogrammeerde microcontroller, maar ook een aantal hardware componenten om hem zo veelzijdig mogelijk te maken. De hardware kan direct met macro instructies worden gestuurd. Het is dus niet nodig om te weten hoe de microcontroller zélf werkt, alles kan in Chip-code worden bestuurd. Toch kunnen we ons voorstellen, dat u meer over de gebruikte controller wilt weten. In [3] is een boek vermeld, dat veel aspecten op een duidelijke manier uitlegt. Ook van het Internet zijn de datasheets te downloaden, bijvoorbeeld van www.vego.nl/chip.
Voor de echte specialisten is het misschien wel interessant om te weten, dat het mogelijk is om door middel van Chip-code de registers in de microcontroller te lezen en te schrijven.

Snel programmeren

Door de efficiënte macro-instructies zijn Chip programma's heel compact en snel te schrijven. Het assembleren van een programma is in seconden gebeurd, waarna het onmiddellijk geladen en getest kan worden. Het aanbrengen van wijzigingen is vrijwel interactief, wat het werken heel plezierig maakt.

De hardware

In figuur 1-1 is het volledig schema van Chip voorgesteld. Centraal daarin staat de microcontroller ST62T65 van ST-Microelectronics (IC2). Het is een CMOS IC zoals ook de andere IC's, om het stroomverbruik laag te houden. De microcontroller bevat een aantal extra componenten, zoals een gewone timer, een auto-reload timer, een analoog/digitaal converter en een seriële perifere interface, kortweg SPI. Alle perifere componenten worden door Chip gebruikt. De gewone timer wordt gebruikt als klok en houdt de seconden, minuten, uren, dagen en weken van het jaar bij. Ook de diverse software timers worden door de gewone timer bestuurd. De auto-reload timer wordt gebruikt om twee servo's onafhankelijk van elkaar te sturen. Beide timers werken op interruptbasis, waardoor Chip in feite "multitasking" is. De afregeling van de klok gebeurt met een software byte, dat door de gebruiker kan worden ingesteld. Eenmaal per minuut wordt de timer met deze waarde gecorrigeerd.
Op K1 zijn vijf digitale uitgangen beschikbaar (Out 0 - Out 4) met serieweerstanden om de microcontroller te beveiligen. Deze uitgangen kunnen worden gezet en teruggezet en de toestand van iedere uitgang kan worden gelezen.



***Figuur 1-1:** Het volledig schema van Chip.*

Op de servo-uitgangen K2 en K3 kunnen normale servo's worden aangesloten, zoals die voor radiografisch bestuurde modellen in soorten en maten verkrijgbaar zijn. De servo's worden onafhankelijk van elkaar door de auto-reload timer gestuurd. Een interruptroutine schakelt via PB6 poort A of poort B van IC1 door.
Op connectorblok K4 zijn vijf ingangen aanwezig. Elke ingang kan digitaal en analoog worden gelezen. In het laatste geval wordt de gekozen ingang met de ADC verbonden. Bij iedere in- en uitgang zijn op de connectors Gnd en V_{dd} aansluitingen aanwezig.
Sensors, LED's en dergelijke kunnen direct op de connectors worden aangesloten. Servo's met Futaba of JR servostekkers passen direct op de servo aansluitingen. Omdat belaste servo's aanzienlijke stromen opnemen, worden ze via een separaat ontkoppelde verbinding vanuit het voedingspunt van de schakeling gevoed.
Voor de communicatie met de gebruiker dient connector K7. Dit is een standaard sub-D connector, die met een verlengsnoer op een seriële poort van de PC wordt aangesloten. Het ingangssignaal wordt door R21, D6 en D7 begrensd en via R13 naar de ingang van de SPI (Sin) gevoerd. Het seriële uitgangssignaal (Sout) bereikt via analoge schakelaar IC3A de uitgangspen van de connector. Als de analoge schakelaar open is, wordt door R18 de uitgang laag gehouden. De niveau's van het seriële uitgangssignaal zijn niet conform de RS232-norm, maar toch werkt deze schakeling betrouwbaar mits het verbindingssnoer niet te lang is. Bij seriële communicatie is PC0 hoog en IC3A gesloten, IC3C en IC3D zijn geopend. Door PC0 laag te maken, wordt de seriële EEPROM (IC4) met de SPI verbonden en via WC (Write Control) schrijfbaar gemaakt. Nu kunnen de software routines via het I^2C-protocol [4] de EEPROM voor lezen en schrijven benaderen. Voor de feitelijke overdracht zorgt de SPI met een klokfrequentie van 308 kHz.
Op connector K8 kan een éénregelig karakter-LCD met zestien letters worden aangesloten. Het is geen écht 16 karakter LCD, maar een 2 x 8 karakter LCD, het meest voorkomende type. Over LCD's kan een heel boek worden geschreven, in [5] staan enkele lezenswaardige publicaties. Voor de aansturing van de databus van het LCD wordt een serieel naar parallel omzetter gebruikt (IC5) zodat voor de overdracht slechts drie uitgangen van de microcontroller nodig zijn. PA5 levert de klok, PA6 de data en PA7 de strobe. De strobe van IC5 is positief, zodat PA7 tevens kan worden gebruikt als (negatieve) strobe om de data in het LCD te klokken. Normaal is PA6 als ingang geschakeld. Op deze ingang is drukknop S1 aangesloten.

ONDERDELENLIJST CHIP BASISPRINT

(zie toelichting in de tekst)

WEERSTANDEN, 1/4 W, 5 %

R1	2,2 Ω
R2	0,22 Ω
R3-R12	1 kΩ
R13,R18	27 kΩ
R14	100 kΩ
R15	120 kΩ
R16	1,5 kΩ
R17	220 kΩ
R19,R20	4,7 kΩ
R21,R22	10 kΩ

INSTELPOTENTIOMETER, PIHER, STAAND, 6 mm

P1	5 kΩ

CONDENSATOREN, RM 2,54

C1,C2	100 µF/16 V staand ø6 x 7 mm
C3,C10-C13	100 nF multilayer
C4,C5	27 pF ceramisch
C6	1 µF/35 V tantaal
C7	4,7 nF ceramisch
C8	4,7 µF/35 V tantaal
C9	470 pF ceramisch

HALFGELEIDERS

IC1	4001
IC2	ST62T65, geprogrammeerd
IC3	4066
IC4	ST24C16
IC5	4094
D1,D3-D7	BAT85
D2	1N4148

DIVERSEN

X1	kristal 8 MHz HC 49/S
SND1	passieve buzzer ø14 x 7,5 RM 7,5 (Elproma type EPM121A1AWP-T of Farnell best. nr. 926-899)
K1,K4	5 x 3-pin male printheader
K2,K3,K6	3-pins SIL male printheader
K5	2-pins SIL male printheader
K7	9-polige haakse sub-D connector, female, voor printmontage
K8	14-polige SIL printheader male
1	IC-voet 8 pens, buscontacten (PreciDip)
2	IC-voeten 14 pens, buscontacten (PreciDip)
1	IC-voet 16 pens, buscontacten (PreciDip)
1	IC-voet 28 pens, buscontacten (PreciDip)

S1	drukknop type D6 (ITT/Schadow)
17	draadbruggen
1	print Chip
LCD	display 1 regel x 16 karakters (Display best. nr. 71.52.1116, Conrad best. nr. 183261, Farnell best. nr. 142-542)

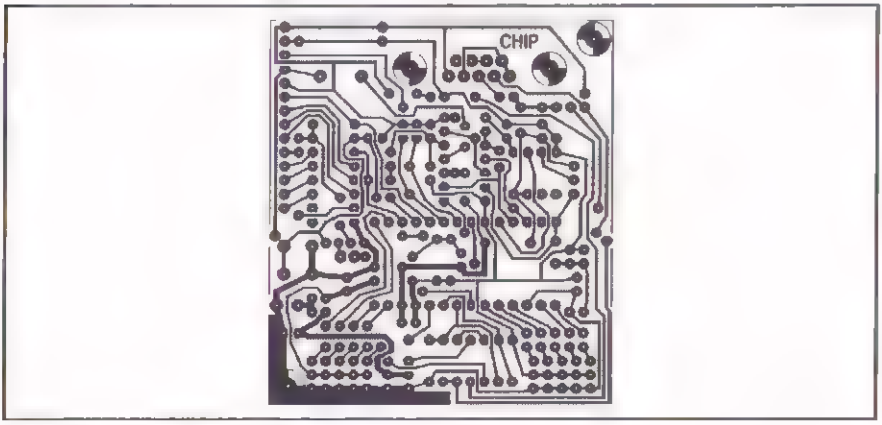
Deze wordt eenmaal per seconde door het operating system getest en als de drukknop is ingedrukt wordt een draaiend Chip-programma gestopt en teruggesprongen naar de commandoprocessor. S1 is dus geen resetknop maar een breakknop en de realtime klok blijft doorlopen. Aansluiting Reg/Sel van het LCD dient voor de keuze karaktermode of commandomode. Deze aansluiting wordt door PB5 gestuurd. Tevens wordt PB5 voor de sturing van sounder SND1 gebruikt. Bij de eerste negatief gaande klok, gebruikt om data voor het LCD in het schuifregister te zetten, wordt analoge schakelaar IC3B geopend en de sounder uitgeschakeld (D3, R16, C7). Na afloop van de data transfer wordt C7 via R15 geladen en IC3B gesloten. Met P1 kan het contrast van het LCD worden ingesteld. Dit hoeft over het algemeen maar één keer te worden gedaan. Als voor het LCD een type met backlight wordt genomen is dat ook bij weinig of geen omgevingslicht goed afleesbaar.
De penbezetting van K8 komt overeen met die van de in de onderdelenlijst genoemde LCD's.
De power-up reset wordt verzorgd door D1, R14 en C6.

De print van Chip

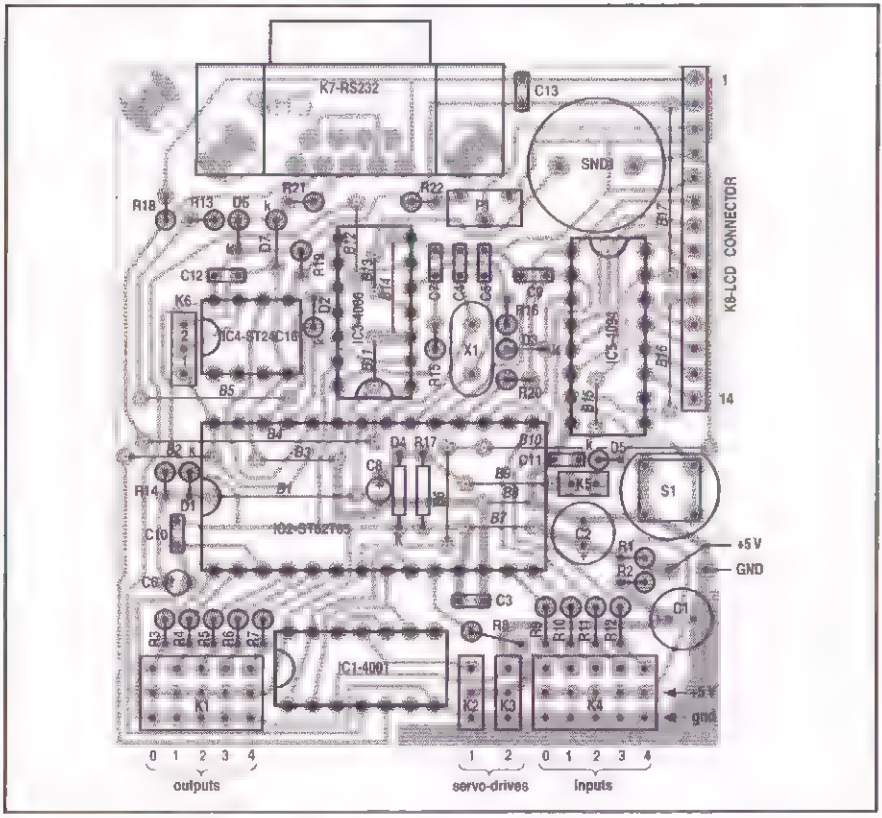
In figuur 1-2 is de **enkelzijdig print** voorgesteld. **Let op:** deze figuur is niet op schaal 1/1! Via de speciale Chip Internet-pagina www.vego.nl/chip kunt u het ontwerp downloaden. Open dit bestand in een beeldbewerkingsprogramma zoals Paint Shop en druk het ontwerp af op transparante folie met als afmetingen 62 x 70 mm².
Boor eerst alle gaten met een boortje van 0,8 mm. Ga nadien de volgende gaatjes uitboren:
- de gaten voor de connectoren en de sounder met 1 mm;
- de vier bevestigingsgaten met 3 mm.

Het bouwen van Chip

De componentenopstelling van de print is voorgesteld in figuur 1-3. Monteer eerst de 17 draadbruggen en breng over B10 ter hoogte van B15 een isolatiekousje aan zodat daar geen kortsluiting kan ontstaan.



Figuur 1-2: *De print van Chip.*



Figuur 1-3: *De componentenopstelling van Chip.*

Gebruik bij voorkeur de aanbevolen voetjes, dan zijn er geen problemen met de draadbruggen. Let op de uitsparing in de voetjes ter indicatie van pen 1. Let op verder op dat, vóór de voet voor IC2 wordt geplaatst, eerst C8, D4 en R17 worden gemonteerd. Leg C9 plat op de print in verband met de hoogte én de brug in het IC-voetje. Voor connectoren K1 en K4 is onder een strip dubbelrijige pinheaders gebruikt en boven een strip enkelrijige, alle connectors behalve K7 zijn male. De connectoren zijn met een figuurzaagje voor metaal van de strip afgezaagd.
Als voedingsbron zijn vier penlight NiCad's in een batterijhouder gebruikt. In het snoer met batterijclip is een aan/uit schuifschakelaar (zelfreinigend) opgenomen.
Het snoer is direct in de voedingspunten van de print gesoldeerd. Door de batterijclip kan het accupakket worden losgenomen om te worden opgeladen.
In plaats van R2 hebben we een tweepolige male header gemonteerd. Uit een glaszekering van 1,2 A hebben we het zekeringdraadje gehaald en op een tweepolige female SIL-header gesoldeerd. Deze "zekering" is op de header op de print gezet.
Met een EEPROM ST24C16 kan op positie 1 van K6 een jumper worden gezet. Dan is de EEPROM normaal tegen schrijven beschermd. Jammer genoeg is bij sommige EEPROM's van ander fabrikaat de reactietijd van de Write Control niet snel genoeg. Dan kan WC niet actief worden gebruikt en wordt vast laag gemaakt door een jumper op positie 2 van K6.
Als de seriële ingang (K7, pen 3) laag is, dan staat op knooppunt R21, D6, D7 circa -0,3 V. Weerstand R13 vormt nu met de inwendige pull-up weerstand van Sin (nominaal 100 kΩ) een spanningsdeler. In het extreme geval dat deze pull-up weerstand een lagere waarde heeft dan ca. 60 kΩ, kan het nodig zijn om R13 iets te verkleinen, zodat het niveau op Sin als laag wordt herkend (max. 0,3 x V_{dd}).
Het is mogelijk om de Chip-interpreter zelfstartend te maken door over K5 een jumper te zetten. Door C8 en D5 wordt dan S1 even "ingedrukt gehouden" als Chip wordt aangezet. Door R17 wordt C8 verder opgeladen en wordt de drukknop vrijgegeven. Hetzelfde resultaat wordt bereikt, zonder de jumper over K7, door S1 tijdens het inschakelen ingedrukt te houden.

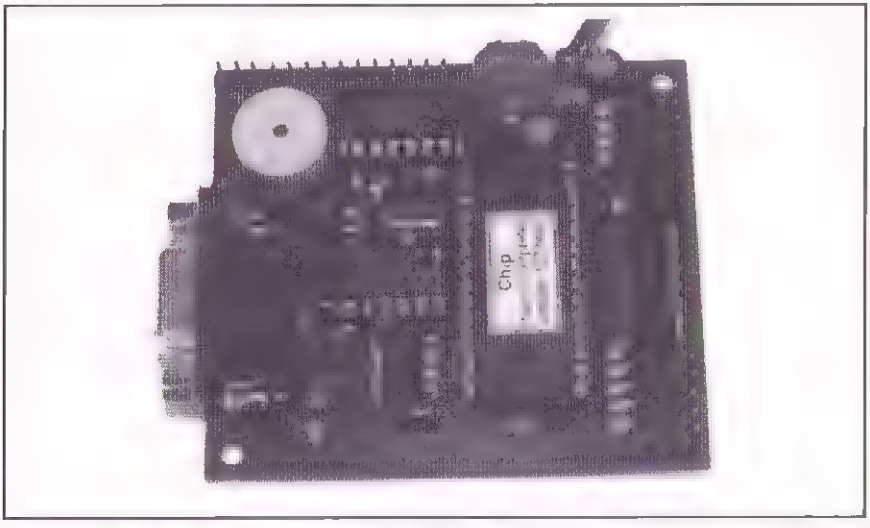
Let op

Van de ST62T65 zijn twee uitvoeringen leverbaar, namelijk met achtervoegsels BB6 en met achtervoegsels CB6 of C. Bij de B-versie is de reset een ingang, maar bij de C-versie een uitgang. Bij de C-versie moet daar-

om resetcondensator C6 worden weggelaten. De reset wordt door de controller zélf verzorgd door de LVD-optie (low voltage detector). Als u de microcontroller zelf programmeert, moet het LVD-bit van de Option Bytes worden gezet en de Option Bytes worden geprogrammeerd. Bij de B-versie hoeft de Option Byte niet persé te worden geprogrammeerd.

Het eindresultaat

Als de print volgens de handleiding is bestukt ziet het eindresultaat er uit zoals voorgesteld in figuur 1-4.



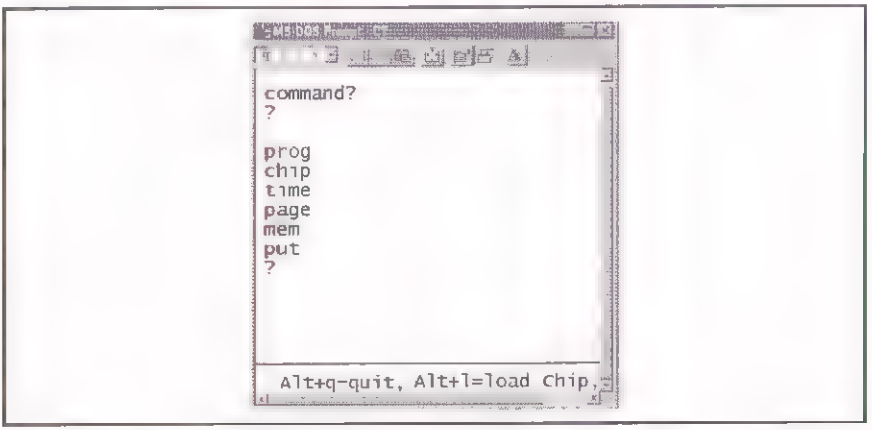
Figuur 1-4: *Het prototype van Chip, dit wijkt iets af van de definitieve versie.*

Het operating system

Chip bevat een eenvoudige commandoprocessor, die via de seriële poort met het terminal programma in de PC communiceert. Het protocol is 19.200,7,n,2. In eerste instantie kan bijvoorbeeld het bij Windows geleverde Terminal programma worden gebruikt, maar om ook programma's naar Chip te uploaden moet **Chipterm.exe** of **Chip VB Terminal** worden gebruikt, zie www.vego.nl/chip.

Voor de ingave mogen uitsluitend kleine letters worden gebruikt. Na ingave van een commando en eventuele parameters, wordt afgesloten met Enter waarna de commandoprocessor in actie komt. Met Backspace kun-

nen fouten worden hersteld en met Escape wordt de hele regel weggegooid. Er zijn zeven commando's, zie figuur 1-5, we zullen ze in vogelvlucht behandelen want het wijst zich eigenlijk vanzelf.



***Figuur 1-5:** De beschikbare commandoset.*

Als het Terminal Programma is gestart en Chip wordt aangezet, verschijnt na een druk op Enter, **command?**, met op de volgende regel de prompt **?**. Door het commando ? te geven (afsluiten met Enter), wordt de commandoset op het scherm gezet.

prog [adres]

Met dit commando kunnen we in de externe EEPROM schrijven en lezen. Alleen de even adressen worden getoond, gevolgd door de bytes op dit adres en het erop volgende. Het laagste adres is 0000 en het hoogste 07FF. Met de + en - toetsen kunnen we het adres verhogen of verlagen. Na de ingave van 2 bytes worden deze weggeschreven en het adres verhoogd. Met Escape keren we terug naar de commandoprocessor.

chip

Dit commando start een Chip programma. De commandoprocessor werkt dan niet meer. Een Chip programma kan ook worden gestart door tijdens het aanzetten van Chip drukknop S1 ingedrukt te houden of op K5 een jumper te zetten. Het programma kan worden gestopt door op de drukknop te drukken, ook de **break** instructie of een fout in het programma doen het programma stoppen.

time

Na ingave verschijnt een lopende 24-uurs klok op het scherm. Drukken op Enter brengt ons in het weekveld, de klok kan worden gelijkgezet. Tweemaal drukken op Escape laat de klok ongewijzigd.

page, mem en put

Met deze commando's kunnen registers in de microcontroller worden gelezen en geschreven.

Adresbereik 00h-3Fh

Eerst moeten we echter iets vertellen over adresbereik 00h-3Fh. De microcontroller heeft behalve normale RAM ook twee EEPROM pagina's en een extra RAM pagina. Met het **page** commando kan een van deze drie pagina's in adresbereik 00h-3Fh worden geprojecteerd.
In EEPROM pagina 0 zetten we van 00h-0Fh een introtekst en op adres 10h de afregelbyte van de klok. Kies eerst met **mem 0** adres 00 van de microcontroller. Dit adres en de bijbehorende byte verschijnen op het scherm, het is een adres in de extra RAM pagina (page 2). Met **page 0** kiezen we EEPROM pagina 0. Met behulp van het commando **put byte** kan byte op dit adres worden geschreven. Met + wordt het adres verhoogd (met - verlaagd). Nu kan de welkom-boodschap in de EEPROM worden gezet:

```
2a, 20, 48, 69, 2c, 20, 49, 27, 6d, 20, 43, 68, 69, 70, 20, 2a
```

Als alle bytes in EEPROM pagina 0 staan, drukken we op Escape om naar de commandoprocessor terug te keren. Als u een lijst met de ASCII-karakters heeft, weet u waarschijnlijk al wat de introtekst is. Om hem op het LCD te zetten moet Chip worden gereset door hem even uit en weer aan te zetten. Op adres 10 van dezelfde pagina zetten we op dezelfde manier de afregelbyte voor de klok. Met de waarde EAh zal hij al vrijwel gelijk lopen. Dus **mem 10, page 0, put ea**.
Verhogen van deze waarde doet de klok langzamer lopen, verlagen sneller. De invloed is pas na 24 uur of langer merkbaar.

Opmerking

Vergeet niet om na iedere aanpassing Chip te resetten, want alleen bij het opstarten wordt de byte uit de EEPROM gelezen, evenals de introtekst.

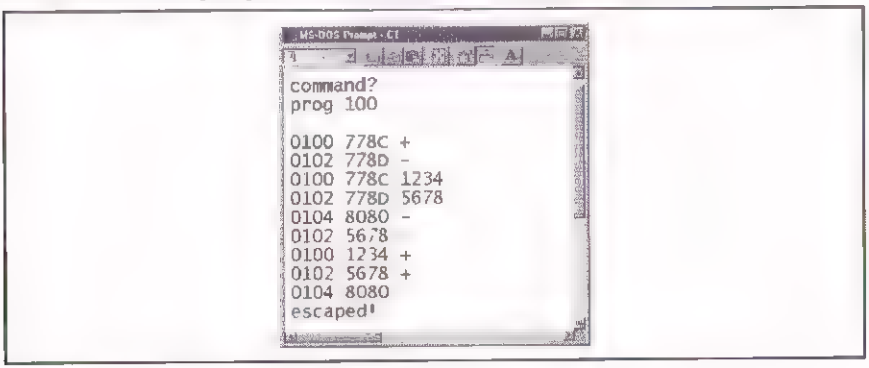
Referenties

[1] Elektuur, september 1999 - april 2000, Basic Stamp Cursus.
[2] Joseph Weisbecker, An Easy Programming System, Byte Publications, december 1978.
[3] Lemmens Luc, ST 62 Microcontrollers, Uitgeverij Elektuur, Postbus 75, 6190 AB Beek.
[4] Het I^2C-protocol staat in de data-sheets van de 24C16. Te downloaden van onder andere www.st.com, www.infineon.com en www.atmel.com.
[4] Een uitgebreide Nederlandstalige toelichting op het I^2C-protocol is als hoofdstuk 6/10.6 verschenen in "Hobby Elektronica & Actueel IC-handboek" (Uitgeverij Vego VOF) aanvulling 47, te bestellen via www.hobbyelektronica.nu
[5] Informatie over LCD's is te vinden op www.epemag.wimborne.co.uk, www.senet.com.au/~cpeacock, www.iae.nl (ga naar piazza en kies pouweha).

2 De Chip instructieset

Inleiding

Een Chip programma bestaat uit een reeks macro instructies, ieder twee bytes lang. Het programma begint op adres 000h van de EEPROM en het hoogste adres van de EEPROM is 7FFh. Een programma kan dus maximaal circa duizend instructies bevatten. Met het commando **prog**, zie figuur 2-1, kan het programma in de EEPROM worden gezet, worden bekeken en/of gewijzigd. Het standaard adres voor prog is 000h en oneven adressen worden niet geaccepteerd, een instructie moet altijd op een even adres staan. Het hoogste adres dat met prog kan worden benaderd is 7FFh, daarboven is het adresgebied echter niet leeg. In het gebied van 800h tot en met 8FFh zijn de registers van de microcontroller geprojecteerd. Die bevinden zich daar natuurlijk niet echt, maar een Chip programma ziet ze daar wel en kan ze lezen en schrijven. Zo is het mogelijk om bijvoorbeeld de auto-reload timer om te programmeren zodat op een van de servo-uitgangen een analoog (PWM) signaal komt te staan.



***Figuur 2-1:** Het commando prog.*

De Chip macrotaal, een eenvoudig concept

Iedere instructie bestaat uit twee bytes, maar het is eenvoudiger ze te beschouwen als zijnde opgebouwd uit vier hex cijfers, ook wel nibbles genoemd. Een nibble bevat vier bits en kan dus de waardes 0h tot en met Fh bevatten. Een Chip instructie bevat vier nibbles en heeft de algemene vorm van:

AXYB

Hierin geeft **A** het **type instructie** aan, **X** en **Y** zijn **variabelen** en **B** is een **precisering** van de instructie. Voor X en Y moeten hex cijfers worden ingevuld, dus een cijfer 0h tot en met Fh, dientengevolge kunnen er zestien variabelen worden geadresseerd. Iedere variabele is één byte groot. Behalve instructies voor variabelen zijn er instructies om de loop van het programma te beïnvloeden, voor conversies, voor het display, enz. Als we de instructieset doorlopen, zie laatste pagina van dit hoofdstuk, wordt alles duidelijk.

De instructieset nader bekeken

In de volgende paragrafen gaan we de voornaamste Chip instructies aan een nader onderzoek onderwerpen.

Program flow

De instructie 0000 (**nop**) doet niets en wordt meestal gebruikt om een of meer plaatsen te reserveren of in software vertragingslussen. Met instructie 1MMM (**jump**) kan naar ieder adres in het programmageheugen worden gesprongen en met 2MMM (**call sub at MMM**) wordt een subroutine op adres MMM aangeroepen. Een subroutine moet altijd worden beëindigd met 00EE (**return from sub**). Er zijn maximaal 8 nesting niveau's mogelijk. Instructie 0050 (**break to operating system**) beëindigt de uitvoering van een Chip programma op een gedefinieerde plaats en brengt de gebruiker terug in de commando processor. Ditzelfde is ook mogelijk door drukknop S1 in te drukken, dan echter is de plaats niet bekend.

Set pointer

Vaak is het nodig om variabelen te converteren van en naar decimaal of hexadecimaal. Dit gebeurt dan op de A-stack (arithmetic stack, rekenkundige stack). Dit zijn drie inwendige microcontroller registers. Door middel van een "pointer", die op alle adressen kan worden "gericht" zijn verplaatsingen (feitelijk kopiëren) van en naar variabelen mogelijk evenals naar het LCD en de A-stack en zelfs naar alle registers in de microcontroller. Omdat de A-stack zo belangrijk is, is er een aparte instructie om de pointer op de A-stack te richten, namelijk **00AA**. MP is de geheugenplaats waar de pointer naar wijst. De instructie **p + 1** zet de pointer 1 positie hoger en de instructie **p + vx** verhoogt de pointer met de waarde van variabele vx. Beide instructies worden als 16 bit optellingen uitgevoerd. Uitsluitend de instructies onder Set pointer kunnen de pointer veranderen.

De instructie **save pointer** slaat een kopie van de pointer intern op en **restore pointer** zet de kopie terug in de pointer.

Pointer conversions on A-stack

Er zijn vier conversie instructies namelijk:
- variabele naar hexadecimaal;
- variabele naar decimaal;
- hexadecimaal naar variabele;
- decimaal naar variabele.

Voor deze conversies wordt de A-stack gebruikt en daarom moet met de instructie **00AA** eerst de pointer daarop worden gezet. Vervolgens kan variabele X naar decimaal worden geconverteerd met **4X13** of naar hexadecimaal met **4X12**. De pointer blijft onveranderd. Stel dat variable A de waarde A7h heeft, dan geeft **4A13** het volgende resultaat:

```
A-stack         31h
A-stack + 1     36h
A-Stack + 2     39h
```

Merk op, dat dit de ASCII-waarden zijn voor respectievelijk 1, 6 en 9, het decimale equivalent voor A7. Omdat het LCD een "ASCII-device" is, kunnen deze cijfers direct op het LCD worden gezet en wel met instructie **DMN3.** Willen we ze op positie 0, 1 en 2 van het LCD, dan wordt de instructie **D023**.
Als we alleen de tientallen en eenheden op het display willen laten zien, kan de pointer met instructie **00A1** (p + 1) op de tientallen worden gezet en de display instructie **D013** kan worden gebruikt.
Als we nu dezelfde variabele A met A7h naar hex willen converteren, dan gebruiken we instructie **4A12** en het resultaat is:

```
A-stack         41h
A-stack + 1     37h
```

Dit zijn de ASCII-waarden voor Ah en voor 7h. We zetten ze op LCD-positie E en F met instructie **DEF3**.
De instructies **4X31** en **4X21** werken precies omgekeerd, waarbij het wel nodig is om eventuele voorlopende nullen op de A-stack te zetten. Dus om bijvoorbeeld het decimale getal 38 te converteren en in variabele C te laden moeten de volgende getallen op de A-stack worden gezet (let op, de maximale waarde van een byte is 255d):

```
A-stack         30
```

```
A-stack + 1    33
A-stack + 2    38
```

Na uitvoering van instructie **4C31** bevat variable C de waarde 26h, het hexadecimale equivalent van 38d.

Pointer-Variable moves

Met de instructies **5XY5** en **5XY7** worden één of meer variabelen gekopieerd in het geheugen vanaf MP, dus het adres waar de pointer naar wijst, en vice versa. De pointer zelf wordt hierbij niet veranderd. X is de begin variabele en Y de eind variabele (X <= Y). Als de pointer naar adres 200h wijst, zal instructie **5665** variabele V6 kopiëren op adres 200h. Omgekeerd zal dan instructie **5667** de inhoud van adres 200h kopiëren in V6. Zo kan iedere individuele variabele dus in het geheugen worden gekopieerd en omgekeerd. Instructie **50F5** zal de variabelen 0-F in het geheugen kopiëren, V0 komt op MP, V1 op MP + 1, enz. Door instructie **50F7** worden de inhouden van de geheugenplaatsen 200h-20Fh gekopieerd in variabelen 0-F ervan uitgaande dat MP nog 200h is. Door middel van deze instructies kunnen variabelen in het programmageheugen worden bewaard en teruggezet. De instructie **5XY7** is uitermate handig om variabelen in één keer vanuit het programmageheugen te initialiseren.

Constants

Hierbij krijgt VX direct de waarde KK (**6XKK**) of wordt KK bij VX opgeteld (**7XKK**). Als de optelling een overflow geeft, dan wordt VF = 01h, anders VF = 00h. Als FFh bij de variabele wordt opgeteld, dan wordt deze (door overflow) effectief met 1 verminderd.

Skip instructies

Uiteraard zijn er ook instructies om condities te testen. Hiervoor dienen de skip instructies. Skip betekent "overslaan", dus als de conditie "waar" is, wordt de erop volgende instructie overgeslagen. Dat zal dan bijna altijd een jump instructie zijn. Het werken met conditionele skips in plaats van met conditionele sprongen is even omdenken, maar na korte tijd is men er volkomen aan gewend. Zo betekent de instructie **9XKK**: skip if VX = KK, dus de instructie **9600** betekent: als variabele 6 de waarde 00h heeft, zal de volgende instructie worden overgeslagen. De instructie **0011** (**skip always**) is handig om een instructie onder te zetten, die in de normale program flow moet worden overgeslagen, maar wél moet worden uitgevoerd als er via een jump naar toe wordt gesprongen. Als **skip always** onder

een conditionele skip wordt gezet, heeft dat tot gevolg dat de conditie wordt omgekeerd, bijvoorbeeld:

```
skip if VX = KK
skip always
jump to MMM
```

heeft tot gevolg dat de sprong wordt uitgevoerd als VX = KK.

Display

Het LCD heeft zestien karakters, het linker is karakter 0h, het rechter karakter Fh. Het display kan direct worden aangestuurd, maar ook indirect, door middel van twee variabelen. De instructie **DMN0** bijvoorbeeld wist het display te beginnen bij karakter M tot en met karakter N. Dus **D330** wist alleen karakter 3 en **D0F0** wist alle karakters van het display. **DXY8** wist het display vanaf het karakter dat in variabele X staat tot dat in variabele Y (X <= Y).
Als de pointer op het begin van een karakterstring wordt gezet, kunnen karakters op het display worden gezet met de instructies **DMN3** of **DXYB**. Het LCD is een "ASCII-device", met andere woorden karakters die op het display worden gezet dienen conform de ASCII-tabel te zijn.
De instructie **DDDD**, "**Rotate text MP on display**" laadt de string naar welks eerste karakter de pointer wijst, vanuit het geheugen in het geheugen van het LCD. De string mag maximaal 24 karakters lang zijn en moet eindigen met 00. Vervolgens wordt door het operating system eenmaal per seconde het commando **shift** naar het LCD gestuurd. Voordat de tekst verschijnt is er een geluidssignaal evenals bij elke shift. Dit is voor de gebruiker een volkomen transparant proces. De tekstrotatie wordt beëindigd door instructie **DDDE**, die het LCD ook reset.

Timers

Chip bevat vier timers, een alarm timer (tone), een timer en een minuten en seconden timer. Alle timers tellen omlaag tot ze nul zijn. De minuten en seconden timer zijn gekoppeld, dus als de minuten timer één lager wordt, dan wordt de seconden timer geladen met 3Bh (59d). Zolang de alarm timer ongelijk nul is, wordt een geluidssignaal opgewekt. De maximale tijdsduur is 9 seconden. Dit is ook de maximale tijdsduur van de timer. Met instructies **CX00** tot en met **CX03** worden de timers in variabele X geladen en met instructies **CX10** tot en met **CX13** worden ze vanuit variabele X geladen.

Clock

De real-time clock hoeft alleen te worden gelezen. Hiervoor dienen instructies **CX04** tot en met **CX08**. Door middel van enkele Chip instructies kan een lopende klok op het LCD worden getoond. Ook is het mogelijk om door middel van een tijdstring in het geheugen te controleren of de kloktijd gelijk is aan de stringtijd en hierop actie te ondernemen. Er kunnen meerdere van deze tijdstrings in het geheugen staan.

Random number

Met instructie **0XKK** wordt vanuit de watchdog timer en de predivider van timer1 een random number (toevalsgetal) gegenereerd. Als masker dient KK. Dit wordt met het random number ge-AND waardoor het random number binnen gebieden kan worden begrensd omdat alleen de bits, die in het masker 1 zijn worden doorgelaten en de overige bits 0 worden. Met masker 07h kan het random number de waarden 0h tot en met 7h krijgen, met masker 0Fh de waarden 0h tot en met Fh, maar met masker 08h alleen de waarden 0h of 8h.
Let op: omdat de instructieset een aantal instructies bevat, waarvan de eerste twee cijfers 00h zijn, mag voor de random number variabele niet V0 worden gebruikt.

Arithmetic

De "A"-groep bevat de logische en rekenkundige bewerkingen. Hierbij zijn altijd twee variabelen betrokken VX en VY, waarbij VX is te beschouwen als de accumulator, dus de variabele die het resultaat zal bevatten (uitgezonderd **AXY0** en **AXY6**). Bij de instructies **AXY4**, **AXY5** en **AXY7** (en ook **7XKK**) wordt variabele VF gebruikt om de carry, borrow of overflow in te zetten. Na deze instructies zal VF 01h zijn bij een carry, borrow of overflow (anders 00h). De instructies **AXY6** en **AXY7** kunnen worden gebruikt om een waarde te "schalen". Door **AXY6** worden VX en VY met elkaar vermenigvuldigd en als 16 bit getal op de A-stack gezet. Instructie **AXY7** deelt dit 16 bit getal door de 8 bit variabele VY. Als het resultaat van de deling groter is dan FFh en daardoor te groot voor de variabele, wordt de overflow gezet. De instructies **AXY6** en **AXY7** zijn heel eenvoudig te gebruiken. Stel dat V0 een waarde bevat die door een temperatuursensor is gemeten en dat deze waarde met 37h/58h moet worden vermenigvuldigd (geschaald) om een waarde in graden Celsius te krijgen:

```
6137          V1 = 37h
6258          V2 = 58h
```

```
00AA            set pointer to A-stack
A016            MP = V0 * V1
A027            V0 = MP/V2
```

Digital I/O

Inwendig bevat Chip 15 uitgangen 0h tot en met Fh, waarvan er 5 (0-4) op connector K1 naar buiten zijn gevoerd. Alle uitgangen kunnen met instructie **E21N** worden geset en met instructie **E20N** worden gereset, waarbij N het nummer van de uitgang is. De stand van iedere uitgang kan door middel van de conditionele skip instructies **E11N** of **E10N** op hoog of laag zijn worden getest. Dit maakt het mogelijk om uitgangen 5h tot en met Eh als bitvlaggen te gebruiken, dus om iets te onthouden. Bitvlag Fh wordt iedere seconde door het operating system gezet. Dat is handig om bijvoorbeeld een real-time klok op het LCD eenmaal per seconde te refreshen (waarbij bitvlag Fh natuurlijk wel moet worden gereset).
De vijf (0-4) digitale ingangen kunnen eveneens door middel van conditionele skips worden uitgelezen.

Analog I/O

De F-groep is voor de analoge I/O. Dit betreft dus het inlezen van een spanning van de ingangen en het sturen van de servo's. Instructie **F3F2** leest van ingang 2 de analoge waarde en zet deze in V3. De maximale waarde is FFh en de minimale waarde is 00h, respectievelijk de spanningen op de V_{dd} en Gnd aansluitingen van de microcontroller.
Instructie **FX0N** doet hetzelfde, maar voegt er een extra bewerking aan toe, namelijk de hoge nibble van VX wordt verplaatst naar de lage nibble en de hoge nibble krijgt de waarde 3. Zo krijgt variabele X de ASCII waarde die overeenkomt met een ingedrukte toets van het Chip keyboard. Als er geen toets is ingedrukt krijgt VX de waarde 3Fh.
Standaard is de servo drive uitgeschakeld, met **FEE1** wordt de pulsopwekking aangezet en met **FEE0** uitgezet. Met **FXE2** en **FXE3** wordt de pulstijd ingesteld voor respectievelijk servo 1 en servo 2.

Chips foutafhandeling

Chip bevat een foutafhandelingssysteem, dat foutieve instructies opvangt. Als een fout wordt ontdekt, wordt op het scherm een foutmelding gegeven met opgave van het adres waar de fout is gevonden. De uitvoering van het Chip programma wordt beëindigd en er wordt teruggekeerd naar de commando processor.

Uitzonderingen

Omdat voor rekenkundige instructies variabele VF als carry, borrow of overflow vlag wordt gebruikt, is het is raadzaam om in een programma de variabele VF niet als normale variabele te gebruiken. Datzelfde geldt voor output F, die eenmaal per seconde door het operating system wordt gezet en die kan worden gebruikt om een bepaalde actie eenmaal per seconde uit te voeren.

Voor het genereren van een random number mag V0 niet worden gebruikt. Dat zou tot onvoorspelbare fouten leiden.

Chip instruction set

Program flow		Mnemonic
0000	nop	nop
1NNN	jump to NNN	jp address [label]
2NNN	call sub at NNN	call address [label]
00EE	return from sub	ret
0050	break to operating system	break

Set pointer		Mnemonic
00AA	set pointer to A-stack	p = a-stack
3NNN	set pointer to NNN	p = address [label]
00A1	pointer + 1	p + 1
4X00	pointer + VX	p + vx
0020	save pointer	save p
0021	restore pointer	rest p

Pointer conversions on A-stack		Mnemonic
4X13	VX -> 3 decimals MP	vx to 3dec mp
4X31	VX = 3 decimals MP	vx = 3dec mp
4X12	VX -> 2 hex MP	vx to 2hex mp
4X21	VX = 2 hex MP	vx = 2hex mp
(Pointer not changed, MP = MSD)		

Pointer-Variable moves		Mnemonic
5XY5	VX...VY -> MP	vx,vy to mp
5XY7	MP -> VX...VY	vx,vy = mp

Constants		Mnemonic
6XKK	VX = KK	vx = kk
7XKK	VX = VX + KK (VF = carry)	vx + kk

Skips		Mnemonic
0011	skip always	skip a
8XKK	Skip if VX <> KK	skip vx <> kk
9XKK	Skip if VX = KK	skip vx = kk
BXY0	Skip if VX = VY	skip vx = vy
BXYF	Skip if VX <> VY	skip vx <> vy
BXY7	Skip if VX < VY	skip vx < vy
BXY8	Skip if VX > VY	skip vx > vy

Display		Mnemonic
DMN0	Clear display from M to N	cd m,n
DXY8	Clear display from VX to VY	cd vx,vy
DMN3	Load display from M to N (from MP)	ld m,n
DXYB	Load display from VX to VY (from MP)	ld vx,vy
DDDD	Rotate text MP on display (string must end with 00, max 24d characters)	rotate
DDDE	stop rotate, clear display	stop rotate

Timers		Mnemonic
CX00	VX = Alarm timer	vx = tone
CX01	VX = Timer	vx = timer
CX02	VX = Seconds timer	vx = sectimer
CX03	VX = Minutes timer	vx = mintimer
CX10	VX -> Alarm timer	vx to tone (1C=1s)
CX11	VX -> Timer	vx to timer (1C=1s)
CX12	VX -> Seconds timer	vx to sectimer
CX13	VX -> Minutes timer	vx to mintimer

Clock		Mnemonic
CX04	VX = Weeks	vx = weeks
CX05	VX = Days	vx = days
CX06	VX = Hours	vx = hours
CX07	VX = Minutes	vx = minutes
CX08	VX = Seconds	vx = seconds

Random number		Mnemonic
0XKK	VX = Random number (KK is .AND. mask)	vx = rnd,kk
Note:	*do not use V0*	

Arithmetic		Mnemonic
AXY0	VX -> VY	vx to vy
AXY1	VX = VX .AND. VY	vx and vy
AXY2	VX = VX .OR. VY	vx or vy
AXY3	VX = VX .XOR. VY	vx xor vy
AXY4	VX = VX + VY (VF = carry)	vx + vy
AXY5	VX = VX - VY (VF = borrow)	vx - vy
AXY6	MP = VX * VY	vx * vy to mp
AXY7	VX = MP/VY (VF = overflow)	vx = mp / vy
Note:	*use the a-stack for AXY6 and AXY7*	

Digital I/O		Mnemonic
E20N	Reset output N	res out n
E21N	Set output N	set out n
E10N	Skip if output N = 0	skip out n = 0
E11N	Skip if output N = 1	skip out n = 1
E00N	Skip if input N = 0	skip in n = 0
E01N	*Skip if input N = 1*	*skip in n = 1*

Notes: *there are 5 inputs (0-4) and 16 outputs from wich 5 (0-4) have pins, 5-E can be used as bitflags. Outputs are default 0*
Output F is set every second by the operating system

```
Analog I/O                                  Mnemonic
FXFN     VX = analog input N                vx = ana n
FX0N     VX = keyboard input N              vx = key n
FEE0     Stop servo drive                   soff
FEE1     Start servo drive                  son
FXE2     VX -> servo 1                      vx to s1
FXE3     VX -> servo 2                      vx to s2
Notes:   digital inputs are switched
         to analog during conversion and the pull-up
         resistor (appr. 200 kohms) is disabled.
         Keyboard values are 30h-3bh, 3fh = no key pressed
         Default servo values are 80h

Chip assembler directives
; This is a comment at the beginning of a line
         org address                        ; comment midline
label    equ address
         bytes 001122....ff
         asciz "Hi, I'm Chip!"
Notes:   an odd number of bytes will be concluded with 00
         asci-zero will be concluded with 00 or 0000

Positioning of characters on display
character 0 ............. character f

Memory model
         08ffh
           |      st62 internal
         0800h
         07ffh
           |      chip programs
         0000h
```

3 Assembler, voorbeelden en keyboard

Chip assembler

De Chip assembler is gemaakt om het schrijven van Chip programma's zo eenvoudig mogelijk te maken. De assembler genereert de macro instructies en berekent de sprongen en pointer posities. De assembler doet dit aan de hand van de mnemonics, die in de Chip instructieset staan. Een mnemonic is een verkorte schrijfwijze van een instructie, die gemakkelijk is te onthouden en die de instructie eenduidig aangeeft. Bij veel instructies wordt een variabele geladen of wordt een kopie van de variabele in een geheugenadres geschreven. Om dit onderscheid eenduidig vast te leggen en ook omdat het gemakkelijk is te onthouden, wordt voor het laden van een variabele de schrijfwijze:

```
vx = ...
```

gebruikt en voor het laden van een adres vanuit een variabele:

```
vx to ...
```

Het Engelse woord "to" betekent "naar", het geeft eenduidig een bestemming aan.

DOS tekstverwerker

Voor het schrijven van Chip programma's in assembly is een tekstverwerker nodig die gewone DOS-tekst maakt, dus er geen opmaakcodes of iets dergelijks tussenvoegt. Programma's als Word of WordPerfect zijn absoluut verboden! Windows Kladblok is bruikbaar, maar een kleine DOS teksteditor, zoals bijvoorbeeld EDT (Norton Editor), is veel handiger. Ook het Windows shareware programma TextPad is zeer geschikt.

Samenstelling assembler code

Een regel in de Chip assemblertaal is als volgt opgebouwd:

```
label       mnemonic      opfield
```

Het **label** dient voor de adresaanduiding, het eerste karakter moet a - z of A - Z zijn. De lengte van een label is maximaal acht karakters. Als een label niet nodig is, moet het eerste karakter van de regel een spatie of een

tab zijn. De **mnemonics** kunnen worden gehaald uit de Chip instructieset. De spaties in de mnemonics mogen eventueel worden weggelaten. Achter de mnemonic kan, na een spatie of een tab, een **opfield** staan. Bij Chip zal dat altijd een adres of een adreslabel zijn.

Assembler aanwijzingen

Er zijn vijf assembler aanwijzingen (directives), die een speciale betekenis voor de assembler hebben.

;

Als het eerste karakter van een regel ; is, dan wordt die regel als een commentaarregel beschouwd. Ook kan na de mnemonic of, indien aanwezig het opfield, na ; commentaar worden opgenomen.

org adres

De tweede aanwijzing is **org adres**. Dit heeft tot gevolg dat het programma(deel), dat op **org** volgt, begint op **adres**. De regel met **org** wordt na assemblage een commentaarregel en mag daarom geen label bevatten.

label equ adres

De derde aanwijzing is **label equ adres**. Hier wordt aan een label een adres toegewezen. Dat is bijzonder handig als met interne registers van de microcontroller wordt gewerkt. In plaats van het lastig te onthouden adres kan de naam van het interne register worden gebruikt. Deze regel wordt ook een commentaarregel in het hex-bestand.

bytes 00112233.....

De vierde aanwijzing is **bytes 00112233.....** Dit heeft tot gevolg dat de bytes 00h, 11h, 22h. 33h, enz. in het programma worden opgenomen. Als de laatste byte op een even adres staat wordt het erop volgende oneven adres opgevuld met 00h.

asciz "Hi I'am Chip!"

De vijfde en laatste aanwijzing is **asciz "Hi I'am Chip!"**. Hier wordt de tekst die tussen de dubbele aanhalingstekens staat omgezet in het ASCII-equivalent in bytes. De assembler sluit de string af met 00 of 0000, om de volgende instructie op een even adres te kunnen zetten. Het sluitkarakter 00 is nodig voor tekst die op het display moet roteren.

Opmerking over bestanden

Het achtervoegsel van een in Chip assembly geschreven programma moet altijd **.asm** zijn, anders wordt het niet door de assembler herkend.

Chipasm.exe

De assembler **Chipasm.exe** (zie www.vego.nl/chip) wordt gestart met **Chipasm naam**. Hierin is "naam" de naam van het te assembleren programma. Het achtervoegsel .asm mag worden weggelaten.
Als door de assembler geen fouten worden geconstateerd, dan wordt een **.hex** bestand gegenereerd, waarvan het voorvoegsel hetzelfde is als van het .asm bestand. Als er wel fouten worden geconstateerd, dan wordt een melding van de aard van de fout en het regelnummer op het scherm gezet.
Het .hex bestand is vrijwel identiek aan het .asm bestand, met dien verstande dat nu de adressen en macro's zijn ingevuld en de regels met org en equ commentaarregels zijn geworden. Als het programma wordt afgedrukt, kan het met commando **prog** in de EEPROM worden gezet of worden ge-upload met het terminalprogramma **Chipterm.exe.**

Opmerking

Het is handig als Chipasm.exe en Chipterm.exe in één map worden gezet, waarin ook de Chip programma's komen te staan. In de map kan ook de DOS-tekstverwerker worden gezet, dat is wel zo makkelijk. Wij hebben Chipasm.exe en Chipterm.exe hernoemd in **CA.exe** en **CT.exe.** Bovendien is de DOS-tekstverwerker (EDT.exe) in een batchbestand (ED.bat) opgenomen, dat het achtervoegsel .asm achter de naam voegt.

Chip VB Terminal

In plaats van het compacte (ca. 50 kB) en snelle DOS-programma **Chipterm.exe** kan ook het Windows-programma **Chip VB Terminal** worden gebruikt (Windows versies 95 en 98). Dit programma is het bij Visual Basic 6 geleverde terminal voorbeeldprogramma, waarvan de tekstverzending is aangepast voor de verzending van Chip hex-bestanden, zie figuur 3-1. De instelling van de vertraging geschiedt met een schuifregelaar in het Properties tabblad. Het programma ziet er mooier uit dan Chipterm, maar voor een functionaliteit die nagenoeg gelijk is, doet het een veel grotere aanslag op de computerbronnen. Na downloaden van www.vego.nl/chip kan het worden geïnstalleerd door het Setup-programma.

Figuur 3-1: Chip VB Terminal tijdens de verzending.

Opmerking

Zowel ChipTerm als Chip VB Terminal hebben de mogelijkheid om .log files te gebruiken. Hierin kan de inhoud van Chip's EEPROM worden gekopieerd (door in **prog** de + toets ingedrukt te houden). Zo kan Chip ook als datalogger worden gebruikt, waarbij de gegevens op de computer kunnen worden verwerkt.

Testfile.asm

Het bestand Testfile.asm (niet afgedrukt in dit boek, alleen te vinden op www.vego.nl/chip) is geen Chip-programma; de inhoud van ieder adres is gelijk aan het adres zodat, voor alle zekerheid, eenvoudig kan worden gecontroleerd of het uploadproces foutloos geschiedt. Bij lange bestanden als Testfile.asm is het mogelijk, dat niet elke upload foutloos is. Probeer het opnieuw als er een checksumfout is.

First.asm

Het programma in listing 3-2 (First.asm) geeft een voorbeeld van de opbouw van een Chip programma en laat het gebruik zien van enkele instructies. Wij gebruiken in Chip programma's uitsluitend onderkast (kleine letters) behalve soms in commentaarregels.
Als het programma is geladen, kan het met CA worden geassembleerd.

CT.exe

Om het hex-bestand in Chip te laden moet CT worden gestart. Als CT voor het eerst wordt gestart, wordt om de te gebruiken COM-poort gevraagd. Deze wordt opgeslagen in het bestand Chipterm.ini.

```
; Listing 3.2: First.asm
; Programma om de call en break instructies te demonstreren,
; het gebruik van de assembler aanwijzingen org en asciz en de
; display instructie ld m,n
;
start       call showtxt        ; call sub showtxt
            break               ; break to op system
;
; subroutine showtxt
;
            org 50              ; start address of sub
showtxt     p - showtx1         ; point p to text string
            ld 0,f              ; load display from char 0 to f
            ret                 ; return from sub
showtx1     asciz "My first program"
;
; end of First.asm
```

Listing 3-2: *De listing van First.asm.*

Belangrijk

In dit bestand staat ook een getal, voor de pauze tussen het versturen van de hex-getallen tijdens het uploaden. De hardware timer van de PC is daarvoor niet fijnmazig genoeg. Pas dit getal aan tot het uploaden zo snel mogelijk gaat, zonder foutmeldingen. Gebruik daarvoor de DOS tekstverwerker. Als tijdens het uploaden een fout optreedt, stopt het uploaden en hangt Chip. Druk op S1 en probeer het opnieuw.
Door in CT de menukeuze ALT+I te kiezen, verschijnt een venster waarin de naam van het .hex bestand kan worden opgegeven. Het achtervoegsel mag worden weggelaten. Na het drukken op Enter wordt het bestand naar Chip gestuurd. Met het commando **Chip** wordt het gestart.

Waar is de poort gebleven?

Soms ziet een in een Windows DOS-venster draaiend programma de seriële poort niet. Neem dan in system.ini onder [386Enh] de aanwijzing:

```
ComNAutoAssigned=0
```

op, waarbij N de gebruikte COM-poort is.

Rechtstreeks adressen wijzigen

We zouden natuurlijk in First.asm de mnemonic **ld 0,f** kunnen veranderen in **rotate** om de tekst te laten roteren, opnieuw assembleren en uploaden, maar het kan ook anders.

Als we het bestand First.hex in de tekstverwerker openen, zien we immers op adres 0052h de instructie **D0F3** staan, waarmee de tekst op het display wordt gezet. Met **prog 52** komen we op dit adres en kunnen direct **DDDD** invoeren en het programma starten. En inderdaad, de tekst roteert nu op het display!

Charset.asm

We willen ook graag weten hoe de ASCII-karakterset van het display er precies uitziet. Vooral de extended karakters interesseren ons, omdat die per LCD kunnen verschillen en we willen weten welk teken we het best voor ° (graden) kunnen gebruiken voor de temperatuurmeting. In listing 3-3 is het programma Charset.asm te zien dat precies dat doet. Eerst zetten we de pointer op de asciz tekst en laden het hele display. Dan wordt de pointer op de A-stack gezet en v0 wordt 00 gemaakt. We gebruiken **output f** om een keer per seconde een aflezing te krijgen, want output f wordt iedere seconde door het Operating System gezet.

```
; Listing 3.3: Charset.asm
; display characterset of the LCD
;
            p = charse2         ; point p to text
            ld 0,f              ; load display
            p = a-stack         ; point p to a-stack
            v0 = 00             ; v0 is counter
charse1     skip out f = 1      ; wait till out f becomes 1
            jp charse1          ; not yet 1, jump back
            res out f           ; out f = 1, reset out f
            v0 to 2hex mp       ; convert v0 t0 2 hex
            ld 5,6              ; load display with hex char's
            v0,v0 to mp         ; copy v0 into mp
            ld f,f              ; and load mp into display pos f
            v0 + 01             ; update counter
            jp charse1          ; jump back into wait loop
charse2     asciz "Byte    is LCD  "
```

***Listing 3-3:** De listing van Charset.asm.*

In een lus wachten we dus rustig tot output f wordt gezet en zodra dat het geval is, wordt de jump geskipt (overgeslagen) en kunnen we aan het werk. Eerst moet natuurlijk output f worden gereset en dan wordt v0 omgezet in twee hex-cijfers, die met **ld 5,6** op positie 5 en 6 op het display worden gezet. Nu kopiëren we v0 naar mp en laden displaypositie f vanuit mp.

Nadat v0 met 1 is verhoogd, is het werk gedaan en springen we terug in de wachtlus.
We assembleren het programma, laden het in Chip en starten het. Bekende, maar ook heel vreemde karakters passeren ons oog. Die zullen wel Japans zijn, want de LCD-controller komt uit Japan. Na even wachten zien we gelukkig een karakter verschijnen dat we prima als gradenteken kunnen gebruiken.
Ons werk heeft zich geloond en we drukken op S1 om naar de commandoprocessor terug te keren!

Labels benoemen?

Weliswaar werkt Charset perfect, maar toch komt misschien de vraag op, waarom de labels geen namen hebben als **wait** en **tekst**? Wel, het is beter om als labelnaam de naam van de (sub)routine te gebruiken in combinatie met een volgnummer (0...9, A...Z).
Vooral in grote programma's, met veel labels, wordt de structuur veel duidelijker en is de kans op fouten door dubbele labels kleiner.

Hexval.asm

Voor veel toepassingen moet de gebruiker gegevens kunnen invoeren als bijvoorbeeld een temperatuur, het aantal NiCad-cellen of het aantal minuten en seconden voor een goed gekookt eitje. We hebben een keyboard nodig. Er is maar één mogelijkheid om het keyboard op Chip aan te sluiten en dat is op een input. Vooraf moeten we testen of dat wel mogelijk is, want anders doen we werk voor niets.
Aan een snoertje met drie gekleurde aders sluiten we een 3-polige female SIL-header aan, voor de verbinding met input 1. We werken de verbindingen netjes af met krimpkousjes want het snoertje zal ons zeker vaker van pas komen. Op de open kant sluiten we een potentiometer van 10 kΩ aan, zodanig dat de spanning op de in-pen regelbaar is. De weerstandswaarde van de potentiometer is niet echt belangrijk, alles tussen 1 kΩ en 100 kΩ is bruikbaar.
Nu komt het programma Hexval.asm (listing 3-4) prima van pas. Ook hier maken we gebruik van de secondenvlag output f, voor een duidelijke weergave op het betrekkelijk langzame display. Voor de eigenlijke meting zorgt een subroutine onder de skip, zodat nu de opdracht **skip out n = 0** moet worden gebruikt.
We assembleren het programma, laden en starten het. Op het display verschijnt een waarde. We verdraaien langzaam de potmeter en wat blijkt,

het is mogelijk alle waarden van 00h tot en met FFh te meten. De proef is prima geslaagd. Het keyboardje moet lukken.

```
; Listing 3.4: Hexval.asm
; show analog voltage on input 0 once a second on the LCD
; display.
; use output flag f, which is set every second by the operating
; system.
;
hexval      p = hexval2         ; point p to display text
            ld 0,f              ; load initial display
            p = a stack         ; point p to a-stack
hexval1     skip out f = 0      ; wait for f to be set
            call hexsub         ; f is set, call hexval sub
            jp hexval1          ; and loop
hexval2     asciz "HEX value =     "
;
hexsub      res out f           ; reset seconds flag
            v1 = 04
            v1 to tone
            v0 = ana 0          ; get voltage from input 0 into v0
            v0 to 2 hex mp      ; convert into 2 hex chars
            ld c,d              ; put 2 hex chars on display
            ret                 ; and return
```

Listing 3-4: *De listing van Hexval.asm.*

Servopot.asm

Nu we toch een potentiometer hebben aangesloten willen we onderzoeken of daarmee een servo kan worden gestuurd. Op **servo 1** sluiten we een Futaba S3003 servo aan, een goedkope en goede servo. In listing 3-5 staat het programma Servopot.asm. Met **son** starten we de servopuls, dan laden we het display met de tekst en zetten de pointer op de A-stack. Dan meten we de potentiometerspanning in v0 en kopiëren v0 naar s1. Als de secondenvlag niet is gezet blijven we in de lus. Als de secondenvlag wel staat, wordt deze gereset, waarna v0 wordt geconverteerd naar 3 decimale getallen. Deze worden op het display gezet en met een jump springen we terug in de lus. Het programma werkt goed. Met de potentiometer kan de servo heel precies worden geregeld. Deze kennis zal zeker van pas komen bij de bouw van de robot.

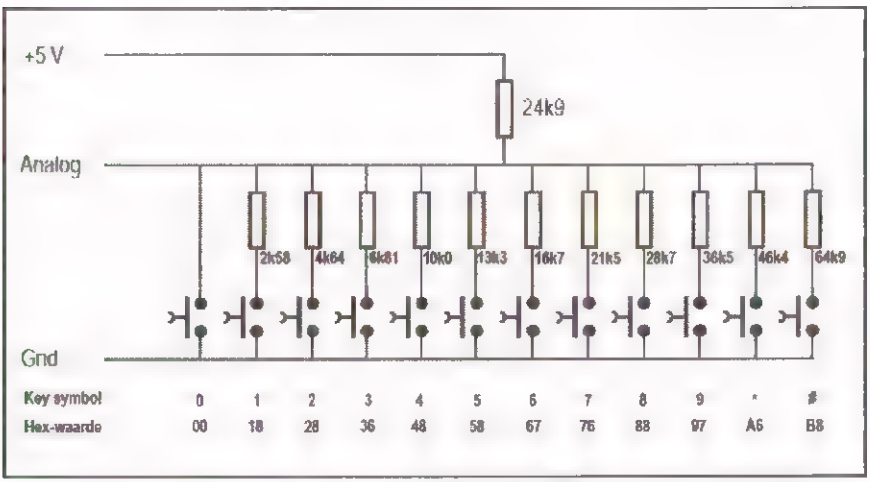
Het keyboard

Als keyboard gebruiken we een type met 12 toetsen (0-9, * en #). De toetsen hebben een gemeenschappelijke aansluiting en iedere toets heeft

een eigen aansluiting. Zo is het mogelijk om iedere toets in een spanningsdeler op te nemen en bij indrukken een eigen, unieke spanning te laten afgeven. De schakeling van het keyboard is getekend in figuur 3-2. In de tabel eronder staan de namen van de toetsen en de hex-waarde van de spanning die bij indrukken wordt gemeten.

```
; Listing 3.5: Servopot.asm
; rotate servo 1 by turning pot connected to input 1
;
            son                 ; start servo drive
            p = servop2         ; point p to text.
            ld 0,f              ; load text to lcd
            p = a-stack         ; point p to a-stack
servop1     v0 = ana 1          ; v0 is voltage from input 1
            v0 to s1            ; load v0 into servo 1
            skip out f = 1      ; skip if seconds flag out f
            jp servop1          ; jump start ofloop
            res out f           ; reset seconds flag
            v0 to 3dec mp       ; convert v0 into 3 decimals mp
            ld 9,b              ; load lcd positions 9,b
            jp servop1          ; jump to start of loop
servop2     asciz "potval = ??? dec"
```

Listing 3-5: *De listing van Servopot.asm.*



Figuur 3-2: *Het schema van het toetsenbord.*

De weerstanden zijn aan de onderkant van het keyboard gemonteerd met aan de achterkant een drie-aderig snoertje met daaraan een connector,

precies zoals het snoertje voor de potmeter. In het schema staan 1 % weerstandswaarden, maar persé nodig is dat niet. Het belangrijkste is dat de lage nibble van iedere toets ligt tussen 5h-Ah, dan is de tolerantie voldoende. Met Hexval.asm in listing 3-4 kan voor iedere toets de weerstandswaarde proefondervindelijk worden bepaald. Bij gebruik van 5 % weerstanden zal het soms nodig zijn om twee weerstanden in serie te schakelen, maar het lukt altijd om een passende hex-waarde te krijgen.

Keytest.asm

Zoals we eerder hebben gezien is Chip "ASCII-georiënteerd", dus moeten ook de toetsen een ASCII-waarde afgeven: 30h tot en met 39h, 3Ah en 3Bh. Dat is nu precies wat de instructie **vx = key n** doet. Als er een toets is ingedrukt, wordt die ontdenderd en nogmaals gemeten. Met het programma Keytest.asm (listing 3-6) kan worden getest of het keyboard goed werkt. Als er een toets wordt ingedrukt wordt er een piepje gegeven en de ASCII-waarde in hex op het display gezet. Dan wordt gewacht tot de toets wordt losgelaten.

Opmerking

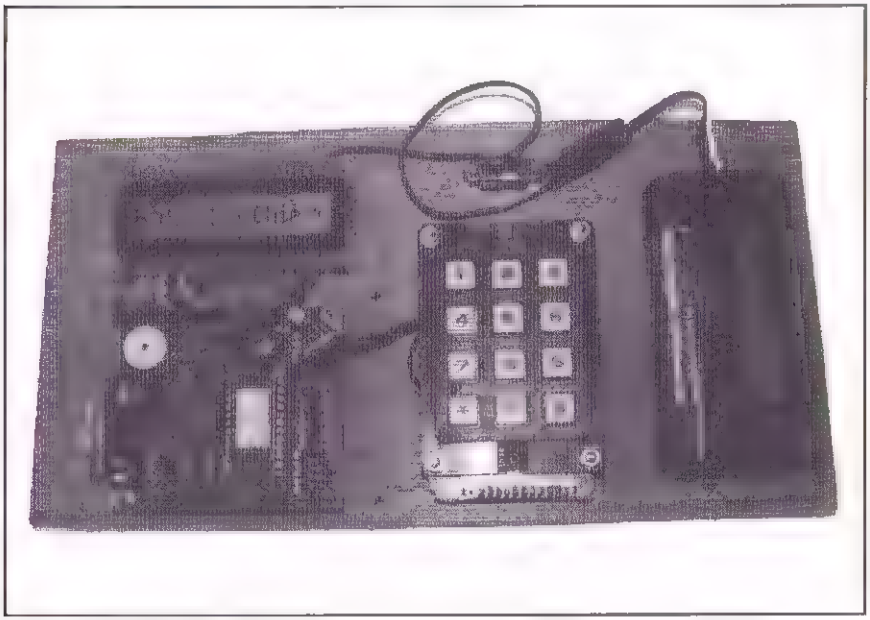
Dit voorbeeld werkt goed, maar er zijn situaties waarbij niet gewacht mag worden omdat de hoofdlus van het programma altijd door moet draaien. Ook dat is op te lossen zoals we in een later stadium zullen zien.

```
; Listing 3.6: Keytest.asm
; test command vx - key n
;
            va   01             ; va is key beep
            p - keytes3         ; load display
            ld 0,f
            p = a-stack         ; point p to a-stack
keytes1     v0 = key 0          ; get key value
            v0 to 2hex mp       ; convert to hex...
            ld c,d              ; ...and put on display
            skip v0 <> 3f       ; skip on key press
            jp keytes1          ; loop
            va to tone          ; give beep
keytes2     v0 - key 0          ; get key value
            skip v0 = 3f        ; skip on no key press
            jp keytes2          ; loop untill key release
            jp keytes1          ; jump for next key press
keytes3     asciz "key value =    h"
```

***Listing 3-6:** De listing van Keytest.asm.*

Experimenteerbord

Ondertussen is het misschien een goed idee om een experimenteersysteem te maken door Chip, het LCD en het keyboard op een plankje te monteren, zie figuur 3-3. Op het plankje kan ook de NiCad-accu een plaatsje vinden. Van dit systeem zullen we veel profijt hebben bij onze verdere ontwikkelingen.



Figuur 3-3: *Het Chip experimenteersysteem.*

Uploaden

Met de 8 bit SPI is een transmissieformaat van maximaal zeven bits mogelijk, omdat bit 7 nodig is als startbit. Alle gegevens worden daarom als hexadecimale karakters naar Chip verzonden. Eerst de vier cijfers van het adres, dan de vier cijfers met de inhoud van dat adres, precies zoals ze in de hex-listing staan. Steeds als er een groep van vier karakters is ontvangen worden deze geconverteerd naar twee bytes en intern opgeslagen. Dan berekent Chip de controlesom en stuurt die naar het terminalprogramma. Na ontvangst van een karakter heeft Chip enige tijd nodig om dat in een buffer te zetten en de SPI opnieuw in te stellen. Het terminalprogramma moet tussen het verzenden van de karakters daarom een korte pauze opnemen in de grootte orde van enkele milliseconden. Deze vertraging wordt verkregen door een for next lus en is afhankelijk van de snelheid van de computer. Chipterm.ini bevat het vertragingsgetal, dat moet worden aangepast (zo klein mogelijk moet worden gemaakt) voor een foutloze transmissie. Het uploaden gaat best wel snel, het schrijven van alle (2.048) EEPROM-cellen duurt ongeveer dertig seconden.

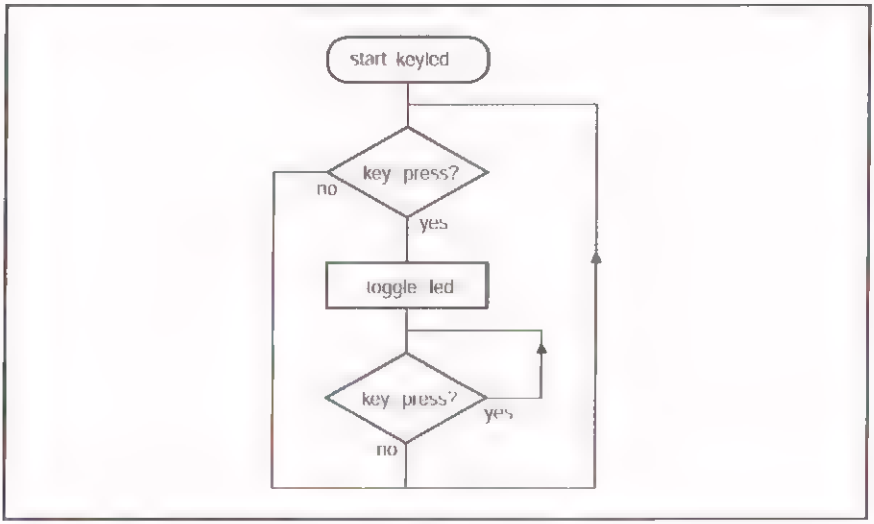
4 Gebruik van het keyboard en een timer

Inleiding

Bij veel programma's moet de mogelijkheid bestaan om gegevens in te voeren. Bij Chip zullen dat altijd getallen zijn, de cijfers 0 tot en met 9. Met het keyboard kan dat en de extra toetsen * en # bewijzen daarbij ook hun nut. Zo kan de * toets bijvoorbeeld als menukeuzetoets worden gebruikt en de # toets als bevestigingstoets. Hiermee kan dan een menusysteem worden gemaakt, bestaande uit meerdere menu's en bij de menu's behorende invoerschermen voor de ingave van de gegevens.
Het inlezen van het keyboard is dus heel belangrijk en daarom beschrijven we twee methodes, de eerste is heel eenvoudig, de tweede is wat ingewikkelder omdat het hoofdprogramma daarbij altijd doorwerkt.

Eenvoudige keyboarduitlezing

Zolang geen toets van het keyboard is ingedrukt, is de uitleeswaarde die **key** geeft 3F. Zodra een toets wordt ingedrukt, wordt de waarde 30-3B, afhankelijk van de ingedrukte toets. Bij iedere volgende uitlezing wordt dezelfde waarde verkregen zolang de toets ingedrukt blijft. Maar we willen voor iedere toetsdruk slechts éénmaal de toetswaarde. Een eenvoudige methode om dat te doen is, om na inlezing en verwerking van een geldige toetswaarde, te wachten tot de toets wordt losgelaten.
In figuur 4-1 is het stroomdiagram te zien van een op deze methode gebaseerd programma.
Direct na de start wordt getest of er een toets ingedrukt is. Als dat het geval is, wordt een LED getoggled, dus aangezet als hij uit is en uitgezet als hij aan is. Daarna wordt gewacht tot de toets wordt losgelaten. Deze methode werkt heel goed zoals na het laden van het programma Keyled.asm (listing 4-1) zal blijken.
Het keyboard is aangesloten op **input 0** en op **output 0** is een LED aangesloten. De 3 mm (low current) LED is op een 3-polige female printheader gesoldeerd, die op output 0 is gezet. De LED kan zo dus op ieder gewenste output worden gezet, wat voor het testen van programma's handig kan zijn.
Bij iedere toetsdruk verandert de LED van toestand. Er wordt direct op iedere toetsdruk gereageerd, maar wat niet is te zien, is dat de programflow stagneert zolang de toets ingedrukt blijft door de sprong naar**waitrel**.



Figuur 4-1: *Het stroomdiagram van de eenvoudige toetsenborduitlezing.*

```
; Listing 4.1: Keyled.asm
; each keypress toggles out 0 on or off
; program waits for key release
;
start     v0 = key 0
          skip v0 <> 3f
          jp continu
;
toggle    skip out 0 = 1
          jp toggle1
          res out 0
          skip a
toggle1   set out 0
;
waitre1   v0 = key 0
          skip v0 = 3f
          jp waitre1
continu   jp start
```

Listing 4-1: *De listing van Keyled.asm.*

Nonstop keyboarduitlezing

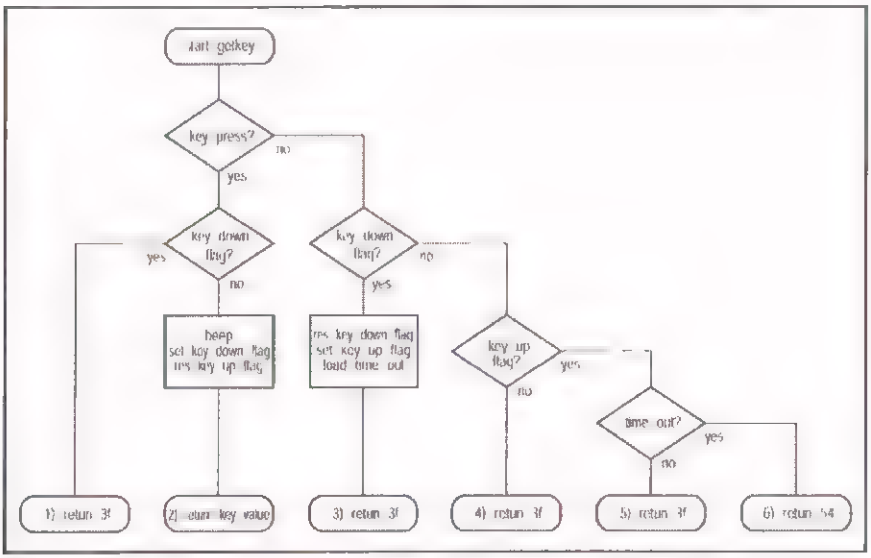
Voor sommige toepassingen is het belangrijk dat een hoofdlus altijd wordt doorlopen. Dat kan bijvoorbeeld bij een acculader zijn waarbij de laadstroom op een bepaalde waarde ingesteld moet blijven. Als dan ook toets-

invoer nodig is, kan niet worden gewacht tot de toets wordt losgelaten. De uitlezing van het toetsenbord mag geen stagnatie geven in loop van de regellus. Vandaar dat deze methode wel "nonstop" wordt genoemd.
In figuur 4-2 is het stroomdiagram van het nonstop keyboard uitleesprogramma Getkey.asm te zien. Het is een subroutine, die altijd wordt doorlopen en een returnwaarde geeft, die afhankelijk is van de toestand van het keyboard. Er worden twee vlaggen gebruikt om de toetstoestand te onthouden. Een dergelijk gebruik van vlaggen wordt ook wel eens vergeleken met een semafoor, een stellage waar vlaggen kunnen worden gehesen, die een bepaalde betekenis hebben.

```
; Listing 4.2: Getkey.asm
; testprogram for getkey subroutine
;
; Remark: press key for some seconds, wait after key release
; some seconds
;
start     P = getktxt
          ld 0,f
          p - a-stack
          vd   80          ; time out value
          ve - 03          ; beep time
;
mainlo    skip out f - 1
          jp mainlo
          res out f
          call getkey
          v0,v0 to mp
          ld f,f
          jp mainlo
;
getktxt asciz "Getkey return:  "
;
; Subroutine getkey
;
; v0 - return value: key press       30...3b
;                    no key press    3f
;                    time out        54
; after first key release the time out is activated
; time out can be disabled by resetting the key up flag: out e
;
getkey    v0 - key 0
          skip v0 <> 3f    ; skip when key is pressed
          jp getkey2       ; no key press, jump
          skip out d - 0   ; skip when key down flag not set
          jp getkey1       ; key down flag is set, jump
          ve to tone       ; give beep
          set out d        ; set key down flag
```

```
          res out e          ; reset key up flag
          ret
getkey1   v0 = 3f            ; key down flag set, return v0 = 3f
          ret
getkey2   skip out d = 1     ; skip when key down flag is set
          jp getkey3         ; key down flag not set, jump
          res out d          ; reset key down flag
          set out e          ; set key up flag
          vd to timer        ; load time out
          ret                ; return with v0 = 3f
getkey3   skip out e = 1     ; skip when key up flag is set
          ret                ; key up flag not set, return with
                             ; v0 = 3f
          v0 = timer         ; key up flag was set, get time out
          skip v0 <> 00      ; skip on no time out
          jp getkey4
          v0 = 3f            ; no time out, return with v0 = 3f
          ret
getkey4   v0 = 54            ; time out, return with v0 = 54 ("T")
          ret
```

Listing 4-2: *De listing van Getkey.asm.*



Figuur 4-2: *Het stroomdiagram van Getkey.asm.*

De **key down flag** wordt gezet als een toets wordt ingedrukt, de **key up flag** als een toets wordt losgelaten. De vlaggen zijn in eerste instantie gereset (outputs van Chip). Uit het stroomdiagram blijkt dat de routine wordt

verlaten via 4) return 3F. Na het indrukken van een toets wordt het programma via 2) verlaten met de waarde van de toets als returnwaarde. Onderweg wordt de alarmtimer geladen voor een keybeep, de key down flag gezet en de key up vlag gereset (wat bij de eerste doorgang al het geval was). Bij de volgende doorloop zal de toets nog ingedrukt zijn en wordt het programma verlaten via 1) ook met returnwaarde 3F. Na verloop van tijd wordt de toets losgelaten en bij de volgende doorloop wordt het programma verlaten via 3) met als returnwaarde weer 3F. Nu worden onderweg de key down vlag gereset, de key up vlag gezet en een time out geladen. Als de toets los blijft en er is geen time out, dan wordt het programma via 5) verlaten met als returnwaarde 3F en als de toets niet tijdig weer wordt ingedrukt, loopt de time out af en wordt het programma via 6) verlaten met als returnwaarde 54.

De waarden 3F en 54

De waarden 3F en 54 zijn eigenlijk willekeurig gekozen. De returnwaarde 3F is de waarde die het keyboard geeft als geen toets is ingedrukt en is de ASCII-waarde voor ?. De waarde 54 is de ASCII-waarde voor T, toepasselijk voor time out.

De routine Getkey

De routine Getkey moet altijd worden doorlopen en zal daarom in een lus zitten. Er zijn daarbij enkele zaken die aandacht vragen. In de eerste plaats moet direct na Getkey de returnwaarde worden geanalyseerd. Zolang die 3F is, is er geen geldige toets, bij 54 is er time out en een andere waarde is een geldige toets. Pas na het indrukken en weer loslaten van een toets wordt de time out gestart. De time out geldt hier dus niet voor de tijd dat een toets ingedrukt blijft. Dat is ook niet nodig, want een ingedrukte toets zal ooit wel worden losgelaten. Als het vereiste aantal toetsdrukken "binnen" is, kan de time out worden uitgezet door de key up vlag te resetten. Na de eerste toetsdruk geeft een "gezette" key up vlag aan dat de toets "los" is.

In listing 4-2 wordt de subroutine Getkey in een eindeloze lus door het hoofdprogramma opgeroepen. De time out waarde en de beeptijd worden vooraf insteld met vd respectievelijk ve. Deze variabelen evenals key down vlag out d en key up vlag out e moeten hier verder maar gereserveerd blijven. De time out en beep time kunnen natuurlijk ook direct worden geladen. Voor de returnwaarde wordt v0 gebruikt. Getkey wordt door het hoofdprogramma eenmaal per seconde aangeroepen, anders zou

een geldige toetswaarde direct worden overschreven met 1) return 3F. Vandaar dat het keyboard ook vertraagd moet worden bediend.

Een timer

In de Eggtimer (listing 4-3) wordt het gebruik van subroutine Getkey gedemonstreerd. Na de start van de timer verschijnt op het display een welkomboodschap met verwijzing naar de * toets. Alleen als deze toets wordt ingedrukt, verschijnt een invoerscherm. Nu kunnen vier cijfers worden ingetoetst, twee voor het aantal minuten en twee voor het aantal seconden. De maximale waarde is 99 m 99 s. Hierbij worden alleen de cijfers 0-9 geaccepteerd. Als de laatste waarde is ingetoetst wordt bij het loslaten van de toets de timer gestart. Als de timer loopt, kunnen ook toetsen worden ingedrukt, maar alleen bij * stopt de timer en verschijnt de welkomboodschap. Het verstrijken van de tijd wordt aangegeven door een langdurige toon en de welkomboodschap.
Op **output 0** kan een LED worden aangesloten, waaraan is te zien dat de hoofdlus altijd wordt doorlopen. In het programma zijn daartoe de instructies **set out 0** en **res out 0** opgenomen. Tijdens het verversen van het display als de timer loopt, licht de LED steeds kort wat feller op (display acties kosten vrij veel tijd), verder brandt hij op halve kracht.

```
; Listing 4.3: Eggtimer.asm
; get a well boiled egg
;
start     p = intrtxt       ; show welcome text on lcd
          ld 0,[
          vc   08           ; set character counter to 08
          vd - 80           ; time out value
          ve   04           ; beep time
          res out c         ; reset eggtimer run flag
          res out e         ; reset key up flag
;
mainloo   set out 0         ; set led on
          skip out c = 0    ; skip when eggtimer not running
          skip out f = 1    ; skip when seconds flag set
          jp mainlo1
          res out e         ; disable time out, key could have
been pressed
          call eggshow
          skip out c = 1
          jp start          ; jump if eggtime is over
mainlo1   res out 0         ; set led off
          call getkey
          skip out e - 0
          jp counter        ; key is released, jump
```

```
          skip v0 <> 3f
          jp mainloo
          skip vc - 08
          jp number          ; counter <> 08, jump to number
          skip v0 = 3a
          jp mainloo         ; first char received <> "*"
;
gotstar   p = eggmask        ; load eggtimer mask on LCD
          ld 0,f
          vc + 01            ; set display counter to tens of
                             ; minutes
          p - inpline        ; set p to input line
          res out c          ; reset eggtimer run flag
          jp mainloo
;
number    v0 + c6            ; add c6 to test for "*", "#"
          skip vf - 00
          jp mainloo         ; char > 39 ("*" or "#"), jump
          v0 + 3a            ; add 3a tot restore original value
          v0,v0 to mp        ; store number
          ld vc,vc           ; show number
          p + 1              ; increment pointer...
          vc + 01            ; ...and character counter
          skip vc <> 0b      ; if charcounter on "m"
          vc = 0d            ; set charcounter to tens of seconds
          jp mainloo
;
counter   skip v0 <> 54      ; 54 = "T"
          jp start           ; time out, jump
          skip vc = 0f       ; skip when 4 numbers have been
                             ; received
          jp mainloo
;
; 4 numbers received, load the eggtimer and set eggtimer run
; flag
;
eggload   res out e          ; reset key up flag to disable time
                             ; out
          vc = 08            ; set charcounter to 08
          p - inpline        ; set pointer to inputline
          v0 = 30            ; hundreds must be set to "0"
          v1,v2 = mp         ; v1 = tens, v2 = units
          p = a-stack
          v0,v2 to mp        ; put minutes decimal number on
                             ; a stack
          v3 = 3dec mp       ; convert tot hex
          v3 to mintimer     ; load minutes timer
          p = inpline
          p + 1
          p + 1
          v1,v2 = mp         ; now get the seconds...
          p = a-stack        ; ...and repeat the same
```

```
          v0,v2 to mp
          v3 = 3dec mp
          v3 to sectimer
          set out c          ; set eggtimer run flag
          jp mainloo
;
; eggshow, routine to show minutes and seconds timer
;
eggshow   res out f
          v0 = mintimer      ; get minutes into v0
          p = a stack
          v0 to 3dec mp      ; convert to decimal on the a stack
          p + 1              ; point to tens
          ld 9,a             ; display tens and units
          v1 = sectimer      ; get seconds into v1
          p = a-stack
          v1 to 3dec mp      ; convert to decimal on the a-stack
          p + 1              ; point to tens
          ld d,e             ; display tens and units
          v0 or v1           ; v0 = v0 .or. v1
          skip v0 = 00       ; skip if both zero
          ret                ; not yet zero, return
          v1 = ff            ; alert user that
          v1 to tone         ; time has run out
          res out c          ; reset eggtimer run flag
          ret
;
; Subroutine getkey
;
; v0 - return value: key press      30...3b
;                    no key press   3f
;                    time out       54
; after first key release the time out is activated
; time out can be disabled by resetting the key up flag: out e
;
getkey    v0 - key 0
          skip v0 <> 3f      ; skip when key is pressed
          jp getkey2         ; no key press, jump
          skip out d = 0     ; skip when key down flag not set
          jp getkey1         ; key down flag is set, jump
          ve to tone         ; give beep
          set out d          ; set key down flag
          res out e          ; reset key up flag
          ret
getkey1   v0 = 3f            ; key down flag was set,
                             ; return v0 - 3f
          ret
getkey2   skip out d = 1     ; skip when key down flag is set
          jp getkey3         ; key down flag not set, jump
          res out d          ; reset key down flag
          set out e          ; set key up flag
          vd to timer        ; load time out
```

```
          ret                ; return with v0 = 3f
getkey3   skip out e = 1     ; skip when key up flag is set
          ret                ; key up flag not set,
                             ; return with v0 = 3f
          v0 = timer         ; key up flag was set, get time out
          skip v0 <> 00      ; skip on no time out
          jp getkey4
          v0 = 3f            ; no time out, return with v0   3f
          ret
getkey4   v0 = 54            ; time out, return with v0 = 54 ("T")
          ret
;
inpline   bytes 00112233     ; storage place for received numbers
intrtxt   asciz "Eggtimer press *"; welcome text
eggmask   asciz "Eggtime: ??m ??s"; input en run time display
```

Listing 4-3: *De listing van Egtimer.asm.*

In het programma zijn vier byteposities gereserveerd voor de opslag van de invoer (**inpline**). Door teller vc wordt de positie op het display bijgehouden. De beginwaarde is 08, juist voor de minuten. Als door Getkey de key up flag is gezet wordt naar counter gesprongen, waar getest wordt op time out. Dan wordt de teller getest op 0Fh, in dat geval zijn de vier cijfers binnen en kunnen de minuten en secondentellers worden geladen. De key up vlag moet worden gereset en de timer run vlag worden gezet. In de hoofdlus mainloo(p) zijn direct onder elkaar twee conditionele skip opgenomen:

```
skip out c = 0
skip out f = 1
jp mainlo1
res out c
```

Alleen als de timer loopt (out c = 1) én de secondenvlag is gezet (out f = 1), wordt res out c bereikt. Res out c is nodig omdat tijdens de timerloop een toets kan worden ingedrukt waardoor de time out wordt geactiveerd.

Uitsluiten van # en *

Bij de invoer van cijfers mogen * en # niet worden geaccepteerd. Dat wordt getest door bij v0 de waarde C6 op te tellen. Een cijfer (30-39) zal geen carry geven (vf = 0). De oorspronkelijke waarde wordt hersteld door 3A op te tellen (een carry speelt nu geen rol meer).

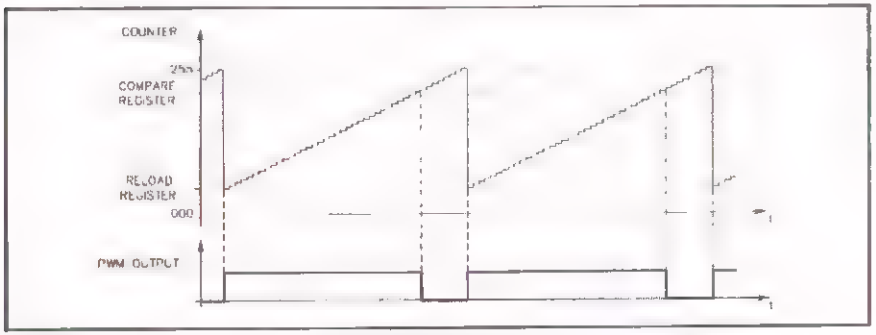
Opmerking

De Eggtimer wordt "onderhuids" door de Chip realtime klok gestuurd en is daardoor op de seconde nauwkeurig.

5 De PWM-timer en een muziekprogramma

Inleiding

De standaard instelling van de PWM-timer is servodriver. Op de uitgangen **servo 1** en **servo 2** kunnen direct twee standaard servo's worden aangesloten, die onafhankelijk van elkaar kunnen worden ingesteld. Servo's zijn er in talloze maten, van uiterst klein tot heel groot en sterk en ze zijn nauwkeurig in te stellen. Bovendien zijn de meeste servo's aan te passen om te worden gebruikt als motor met vertraging. Soms is beweging niet nodig, maar wel een regelbare spanning. Te denken valt bijvoorbeeld aan een capaciteitsmeter voor NiCad's. Daarvoor is het nodig dat de ontlaadstroom op een constante waarde wordt gehouden. De tijd die nodig is tot de accu tot is ontladen, vermenigvuldigd met de ontlaadstroom, bepaalt de capaciteit van de accu.
Met de PWM-timer kan zo'n regelbare spanning worden opgewekt. In figuur 5-1 is het principe te zien waarop de PWM-timer van Chip functioneert.



Figuur 5-1: *Het principe van de PWM-timer.*

Het is een 8 bit teller, die omhoog telt. Op de teller is een set-reset flip-flop aangesloten. Als de teller zijn eindwaarde (FFh) passeert, wordt de flip-flop geset en tevens wordt de teller geladen met de waarde die in het reload register staat. Vanaf deze waarde telt de teller weer omhoog en bij het passeren van de compare value wordt de flip-flop gereset. De compare value staat in een apart register, het compare register, en het zal duidelijk zijn dat de compare value niet lager mag zijn dan de reload value, want dan wordt de flip-flop niet gereset en blijft hoog.

Op de uitgang van de flip-flop staat het PWM-signaal. De inhoud van het reload register bepaalt dus de frequentie van het PWM-signaal en de waarde van het compare register de werkslag. De resolutie van het PWM-signaal is het hoogst als het reload register de waarde 00h bevat. Het compare register mag dan alle waarden van 00h-FFh bevatten. Op de servo uitgangen van Chip is het PWM-signaal geïnverteerd aanwezig en aan het begin van de "PWM-ladder" is het signaal dus laag. Daardoor krijgen we tijd voor de afhandeling van de interrupt routine die de servo's afwisselend aanstuurt. Voor de opwekking van een analoog signaal is geen interrupt routine nodig en kan het hele tellerbereik worden gebruikt.

PWM-sturing met Chip

Nu we weten hoe de PWM-timer werkt, rijst de vraag hoe we daar in Chip programma's gebruik van kunnen maken. Daarvoor is het nodig dat we een excursie maken naar het inwendige van de microcontroller, namelijk de registers. Angsthazen hoeven het nu niet op een lopen te zetten, want de programma's in de listings kunnen direct in eigen toepassingen worden opgenomen.

ARMC (adres D5h), AR-mode control register

Door hierin 00h te zetten stoppen we de PWM-timer. Met E0h werkt de timer in PWM-mode.

ARRC (adres D9h), AR-reload capture register

Voor een PWM-signaal wordt dit met 00h geladen.

ARCP (adres DAh), AR-compare register

De waarde hierin (00h-FFh) bepaalt de werkslag.

ARSC1 (adres D7h), AR-status control register 1

De waarde hierin bepaalt de frequentie van het PWM-signaal volgens tabel 5-1. Alle frequenties kunnen nog met een factor 3 worden verlaagd doordat met bit 0 een extra delertrap kan worden ingeschakeld. De lage nibble wordt dan 1h in plaats van 0h. **Let op:** de frequentie mag alleen worden veranderd als de timer stilstaat!

DRB (adres C1h), data register van port B

Met bit 6 van dit register kan het PWM-signaal naar **servo 1** of **servo 2** worden gestuurd. Op dit register zijn ook de uitgangen van Chip aangeslo-

ten en we moeten hier zeer behoedzaam te werk gaan. De truc is als volgt. Eerst laden we de waarde van drb in een Chip variabele. Vervolgens maken we met behulp van een .AND. masker (3Fh) bits 6 en 7 nul. Dan maken we bit 6 naar keuze 0h of 1h en bit 7 altijd 1h met een .OR. masker (80h of C0h). Tenslotte schrijven we het resultaat terug in drb.

ARSCR1 byte	deel-factor	PWM-frequentie
00	1	32.250 Hz
20	2	16.125 Hz
40	4	8.063 Hz
60	8	4.031 Hz
80	16	2.016 Hz
A0	32	1.008 Hz
C0	64	504 Hz
E0	128	252 Hz

Tabel 5-1: *Deelfactoren voor de PWM-timer.*

Demonstratie programma

In listing 5-1 staat het demonstratie programma PWMtimer.asm voor de PWM-timer. Eerst worden de register gedeclareerd met de **equ aanwijzingen**. De registers van de microcontroller zijn geprojecteerd in adresgebied 800h-8FFh. Dan volgt de hoofdlus van het programma waarin eerst een tekst op het display wordt gezet. Nu wordt de subroutine **arstart** opgeroepen, die de PWM-timer start. De variabelen v0-V5 worden geladen met de waarden uit de bytes string. Dan laden we ARMC met 00h om de timer te stoppen voor het geval hij liep. Nu is v0 vrij gekomen en wordt geladen met de waarde van DRB. Bit 7 en 6 worden 1h gemaakt en v0 wordt teruggeschreven naar DRB. Vervolgens worden de overige registers geladen en de timer gestart.
In ARSC1 wordt de waarde A0h geschreven voor een deelfactor van 32. De PWM-frequentie bedraagt dan circa 1 kHz. In de lus van het hoofdprogramma **pwmmain** wordt v0 geladen met de analoge spanning op ingang 1. De waarde wordt omgezet naar twee hex-cijfers, die op het display worden gezet. De subroutine **arload** tenslotte zet de waarde in ARCP.
Met een potentiometer, die is aangesloten op ingang 1, kan de werkslag van het PWM-signaal worden ingesteld van 0 %-100 % (00h-FFh). De eindwaarden worden niet helemaal bereikt, er blijven korte "spikes" aanwezig. Als een zuivere gelijkspanning nodig is, kan op de PWM-uitgang

een RC-filter als integrator worden aangesloten. Als we met bijvoorbeeld een kristal oortelefoontje het PWM-signaal afluisteren, valt op dat bij verandering van de werkslag weinig verandert aan de toonhoogte. Wel wordt bij de uiterste standen het geluid steeds zwakker.
Met subroutine **servo2** in dezelfde listing kan het PWM-signaal naar servo 2 worden geleid. De oproep voor de subroutine komt dan onder **call arstart** in het hoofdprogramma te staan. Met de drie subroutines kan de PWM-timer in Chip programma's worden gestuurd. De PWM-timer kan ook worden stilgezet met de Chip opdracht **soff** en weer in servomode worden gestart met **son**.

```
; Listing 5.1: PWMtimer.asm
;
; artimer (pwm timer) register adressen and port b data register
drb         equ 8c1          ; port b data register
arsc1       equ 8d7          ; ar status control register 1
arrc        equ 8d9          ; ar reload register
arcp        equ 8da          ; ar compare register
armc        equ 8d5          ; ar mode control register
;
; pwmmain, main routine for pwm demonstration program
;
pwmmain     p = dutytxt      ; point to dutycycle text and..
            ld 0,f           ; load display
            call arstart     ; start artimer in pwm mode on
                             ; servo 1
pwmloop     v0 = ana 1       ; v0 = analog value from input 1
            p = a-stack      ; point to a stack
            v0 to 2hex mp    ; convert v0 to 2 hex
            ld d,e           ; show on display
            call arload      ; set duty cycle (v0 will be
                             ; inverted)
            jp pwmloop       ; and do it again
;
dutytxt     asciz "dutycycle  = ?? "
;
; arstart, subroutine to start artimer in pwm mode
;
arstart     p = pwminit      ; point to pwm initialize values
            v0,v5 = mp       ; load into v0...v5
            p = armc         ; point to ar mode control register
            v0,v0 to mp      ; stop artimer, it could be running
            p = drb          ; point to port b dataregister
            v0,v0 = mp       ; v0 = port b data register
            v1 or v0         ; we don't want to change the
                             ; outputs, so we .or.
            v1,v1 to mp      ; select servo 1 (11?? ????b)
            p = arsc1        ; point to artimer status
```

```
                              ; control register 1
          v2,v2 to mp         ; load predivider ratio (a0h = :32)
          p = arrc            ; point to ar reload register
          v3,v3 to mp         ; load reload value (00h)
          p = arcp            ; point to ar compare register
          v4,v4 to mp         ; load ar compare value (ffh)
          p = armc            ; point to ar mode control register
          v5,v5 to mp         ; start pwm generation
          ret
pwminit   bytes 00c0a000ffe0
;
; arload, subroutine to invert received value (v0) and load ar
; compare register
;
arload    v1 = ff             ; v1 = xor value 1
          v0 xor v1           ; v0 = value received, invert by
                              ; .xor.
          p = arcp            ; point to ar compare register and..
          v0,v0 to mp         ; load v0 into ar compare
          ret
;
; servo2
;
servo2    p = drb             ; point to port b dataregister
          v0,v0 = mp          ; v0 = port b data register
          v1 = 3f             ; v1 is .and. mask 3f
          v0 and v1           ; reset bits 7 and 6
          v1 = 80             ; v1 is .or. mask 80
          v0 or v1            ; set bit 7
          v0,v0 to mp         ; write back to drb
          ret
```

***Listing 5-1:** Het demonstratie programma PWMtimer.asm.*

Musicbox

Met de PWM-timer kunnen ook tonen worden opgewekt. Maar omdat dan de frequentie veranderbaar moet zijn, moet het compare register een vaste waarde hebben en de waarde van het reload register bepaalt dan de toonhoogte.
Weliswaar verandert de werkslag van het PWM-signaal, maar dat is bij deze toepassing niet erg. Het geeft het geluid zelfs een eigen karakter. Door de waarde van de prescaler aan te passen is het mogelijk om meerdere octaven te bestrijken. De timer moet in de autoreload mode zijn ingesteld. Om het geluid hoorbaar te maken kan op de **servo 1** uitgang een kristal oortelefoontje of een passieve piëzo sounder (zoals op de print van Chip) worden aangesloten.

Voorbeeld

De volgende toepassing is ontleend aan de ST626x family starterkit en aangepast voor Chip. De melodietjes zijn ongewijzigd. Het concept is eenvoudig maar heel aardig. Degenen, die een beetje muzikaal zijn, kunnen makkelijk andere melodietjes maken.
Eerst bepalen we de waarden waarmee het reload register moet worden geladen voor de zeven hoofdtonen in een octaaf: DO, RE, MI, FA, SOL, LA en SI. Het compare register is vooraf geladen met F0h.
Om het einde van de melodie aan te geven wordt in plaats van de waarde van de toon 00h gebruikt, zie tabel 5-2.
Voor vier octaven zijn de waarden waarmee het prescaler register (arsc1) moet worden geladen als gegeven in tabel 5-3.

DO	60h
RE	71h
MI	81h
FA	88h
SOL	95h
LA	A0h
SI	ACh
Einde	00h

Tabel 5-2: *De toonwaarde voor de verschillende noten.*

Octaaf 1	C1h
Octaaf 2	A1h
Octaaf 3	81h
Octaaf 4	61h

Tabel 5-3: *De waarden in het prescale register voor vier octaven.*

Tot slot is het nog nodig om de lengte van de tonen in te kunnen stellen. Daarvoor gebruiken we de timer van Chip met de in tabel 5-4 aangegeven waarden voor respectievelijk een halve, een hele en een dubbele toon.

Half	04h
Heel	08h
Dubbel	10h

Tabel 5-4: *De waarden voor respectievelijk een halve, hele en dubbele toon.*

Kleine wachtlus

Als tussen de tonen een hele korte pauze zit zonder geluid klinkt de melodie beter. Hiervoor gebruiken we een software wachtlus waarvoor we een waarde 07h gebruiken. De melodietjes worden nu als een tabel opgebouwd en wel als volgt:

```
byte1 = toon, byte2 = octaaf, byte3 = duur, byte4 = pauze
```

Een melodie tabel met als begin het label start komt er dan uit te zien zoals in tabel 5-5 voorgesteld.

```
start   bytes 88A10407 ;  FA   octaaf 2 half      pauze
        bytes 81A10407 ;  MI   octaaf 2 half      pauze
        bytes 88A10407 ;  FA   octaaf 2 half      pauze
        bytes 71811007 ;  RE   octaaf 3dubbel     pauze
        bytes 00       ;  Einde
```

***Tabel 5-5:** De melodie tabel.*

De eenvoudigste methode om zelf melodie tabellen voor de Musicbox te maken is om de noten eerst in symbolische vorm in de tekstverwerker in te voeren, dus zoals na de puntkomma in bovenstaande tabel. Ze zijn dan nog gemakkelijk te lezen. Met de vervangfunctie van de tekstverwerker kunnen de symbolen worden omgezet in hun corresponderende bytes. Het enige dat dan nog nodig is, is de bytes te ordenen zodat de assembler ze slikt.
De pauze had ook als vaste waarde in het programma kunnen worden opgenomen. Dat zou de tabel niet korter maken omdat alle instructies op een even adres moeten staan. De assembler zou aan iedere byte string een 00h-byte hebben toegevoegd.

Musicbox.asm

In het programma Musicbox (listing 5-2) worden eerst de gebruikte interne register adressen van de controller gedeclareerd. Het is duidelijker om met de namen dan met de adressen te werken. Dan wordt de pointer op de melody tabel gezet en gesaved. Vervolgens wordt de pointer op een tabel met instelwaarden voor registers gezet die in één keer in de variabelen v6-va worden ingelezen. Dan wordt de pointer successievelijk op de verschillende registers gezet, die dan vanuit de variabelen worden geïnitialiseerd. Door drb met 80h in plaats van met C0h te laden, kan **servo 2** als PWM-uitgang worden gekozen (Chip's uitgangen worden hier niet ge-

bruikt!). De programmalus begint bij label **next**. Hier wordt de pointer gerestored en een vier bytes regel uit de melody tabel wordt in de variabelen vb-ve gelezen. Dan wordt de pointer op de volgende regel gezet (p + va, va = 04h) en gesaved. Als de toon 00h is wordt het programma afgebroken, anders worden het reload register (toon), het prescaler register (octaaf) en de Chip-timer geladen. Dan wordt de timer aangezet door E0h in het mode control register te laden vanuit v9. De timer wordt in vd geladen en zolang deze niet 00h is, wordt gewacht. Dan wordt de timer gestopt door vanuit v7 de waarde 80 in het mode control register te laden. Tot slot volgt de software delay en wordt naar next gesprongen.

```
; Listing 5.2: Musicbox.asm
; plays music from table, using artimer (pwm-timer)
;
; artimer register adressen and port b data register
;
arscl     equ 8d7          ; ar status control register 1
armc      equ 8d5          ; ar mode control register
arrc      equ 8d9          ; ar reload register
arcp      equ 8da          ; ar compare register
drb       equ 8c1          ; port b data register
;
          p = melody       ; point to start of music table
          save p           ; save pointer
          p - initial      ; point to table with intialize
                           ; values and..
          v6,va = mp       ; load initialize values into v6...va
          p - drb          ; point to drb and..
          v6,v6 to mp      ; connect servo 1 to pb7 and reset
                           ; servo 1
          p = armc         ; point to artimer mode control
                           ; register and..
          v7,v7 to mp      ; set artimer = off and pwm = off
          p= arcp          ; point to artimer compare register
                           ; and..
          v8,v8 to mp      ; load with f0
next      rest p           ; point into music table
          vb,ve = mp       ; vb = note, vc = octave, vd =
                           ; duration, ve = delay
          p + va           ; point to next music table entry
          save p           ; save pointer
          skip vb <> 00    ; skip if note <> 00
          break            ; break if note = 00
          p = arrc         ; point to ar reload register..
          vb,vb to mp      ; load note
          p = arscl        ; point to ar status control
                           ; register 1 and..
          vc,vc to mp      ; load octave into predivider
```

```
            vd to timer      ; load duration into timer
            p = armc         ; point to artimer mode control
                             ; register and..
            v9,v9 to mp      ; set artimer = on and pwm - on
playing     vd = timer       ; wait for duration of note
            skip vd = 00
            jp playing
            v7,v7 to mp      ; set artimer = off and pwm = off
delay       ve + ff          ; wait for delay time between notes
            skip ve = 00
            jp delay
            jp next
;
initial     bytes c080f0e004 ; presets for v6...va
;
melody      bytes 88a10407
            bytes 81a10407
            bytes 88a10407
            bytes 71811007
            bytes 00
;
hymne       bytes 81a10807
            bytes 81a10807
            bytes 88a10807
            bytes 95a10807
            bytes 95a10807
            bytes 88a10807
            bytes 81a10807
            bytes 71a10807
            bytes 60a10807
            bytes 60a10807
            bytes 71a10807
            bytes 81a10807
            bytes 81a11007
            bytes 71a11007
            bytes 81a10807
            bytes 81a10807
            bytes 88a10807
            bytes 95a10807
            bytes 95a10807
            bytes 88a10807
            bytes 81a10807
            bytes 71a10807
            bytes 60a10807
            bytes 60a10807
            bytes 71a10807
            bytes 81a10807
            bytes 71a11007
            bytes 60a11007
            bytes 00
;
stars       bytes 60a11007
```

```
            bytes 95a11007
            bytes 88a10407
            bytes 81a10407
            bytes 71a10407
            bytes 60811007
            bytes 95a11007
            bytes 88a10407
            bytes 81a10407
            bytes 71a10407
            bytes 60811007
            bytes 95a11007
            bytes 88a10407
            bytes 81a10407
            bytes 88a10407
            bytes 71a11007
            bytes 00
;
auclair     bytes 60a10807
            bytes 60a10807
            bytes 60a10807
            bytes 71a10807
            bytes 81a11007
            bytes 71a11007
            bytes 60a10807
            bytes 81a10807
            bytes 71a10807
            bytes 71a10807
            bytes 60a11007
            bytes 00
;
beet        bytes 88a10407
            bytes 88a10407
            bytes 88a10407
            bytes 71a11007
            bytes 00
```

Listing 5-2: *De listing van Musicbox.asm.*

Melodietjes

In de listings staan nog wat andere melodietjes. Door de pointer op het label van de melodie te zetten, wordt deze afgespeeld na het starten van de Chip interpreter. De bij de labels behorende adressen kunnen in de hex-listing worden opgezocht en met **prog** in de instructie op adres 0000 worden gepatched.

Uitbreidingen

Het programma zou kunnen worden uitgebreid met een keuzemenu voor de melodie waarbij dan de naam op het LCD wordt gezet. Ook kunnen met behulp van een lus alle melodieën na elkaar ten gehore worden gebracht.

Elektronische deurbel

Als Chip zelfstartend wordt gemaakt, staat niets de toepassing als deurbel in de weg. Een extra transistor en een luidspreker zijn dan nodig, want als deurbel heeft het oortelefoontje niet voldoende volume.

6 Chip als robot

Opmerking

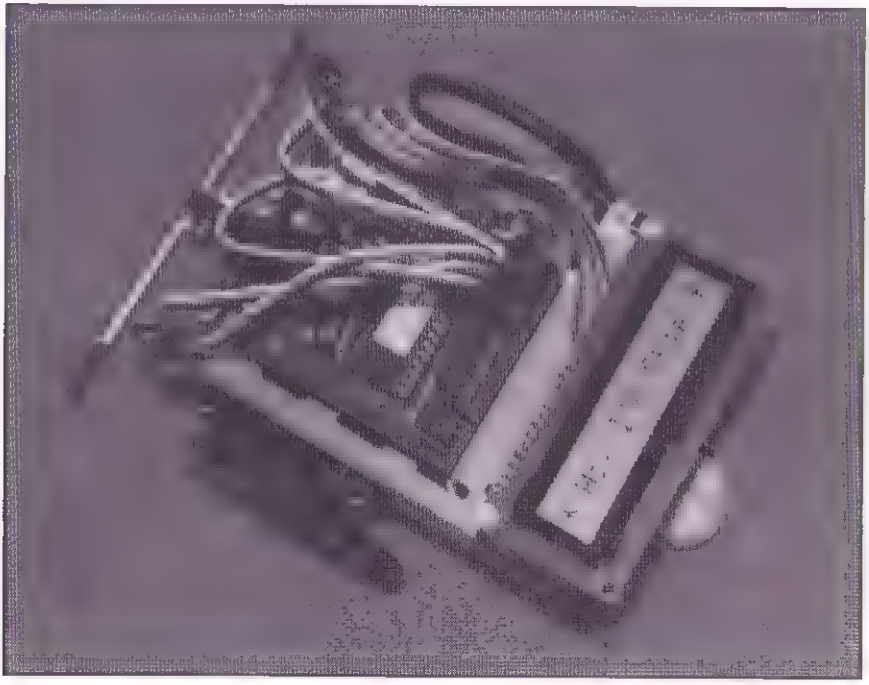
De robot die in dit hoofdstuk wordt beschreven is gebaseerd op het robotwagentje dat ook in de Basic Stamp Cursus wordt gebruikt[1].

De mechanische opbouw

In het chassis zijn twee servo's gemonteerd, die eerst zijn aangepast om continu te kunnen draaien. Door ons zijn servo's FS100 gebruikt. Die kosten vrij weinig en zijn toch goed. De terugmeldingspotentiometer wordt door middel van een plastic koppelstukje in de uitgaande as aangedreven. Door dit koppelstukje te verwijderen en de blokkeringspallen van de as af te snijden wordt de servo een motor.
De stekkers van de servo's worden op de **servo** uitgangen van Chip gezet.
Als achterwiel is een glad rond houten bolletje met een diameter van 25 mm gebruikt, dat is gekocht in een hobbywinkel. Bovenop, aan de voorkant van het chassis, is de Chip computer gemonteerd, met de RS232-connector naar voren wijzend. Achterop het chassis staat, op afstandbussen van 30 mm lengte, het LCD. Onder het chassis, tussen de servo's, hangt het accupakket (vier NiCad penlites in een houder) dat met een kabelbandje vastzit. Het accupakket is zo geplaatst dat de druk op het achterwiel niet hoog is en dit gemakkelijk zijwaarts kan schuiven. Tegen de achter zijkant van de robot is, met dubbelzijdig plakband, de aan/uitschakelaar bevestigd.
De mechanische opbouw van de robot is voorgesteld in figuur 6-1.

Obstakel detectie

Voor de detectie van obstakels is aan de voorkant van het chassis een bumperschakelaar gemonteerd en voorop het chassis zijn links en rechts infraroodsensors (zie verder) opgenomen.
Voor de bumperschakelaar is een microswitch gebruikt met aan de bedieningshevel een stuk messing buis (diameter 3 mm, lengte 100 mm). Over de einden zijn stukjes rubberslang geschoven, die aan beide zijden ongeveer 1 cm buiten het wagentje uitsteken. De bumper zit 20 mm boven de grond en 40 mm voor de voorkant van de robot. Door de constructie is de bumper niet star.



Figuur 6-1: *De robot is volledig opgebouwd uit standaard onderdelen en is door een knutselaar snel in elkaar te zetten.*

Bij een botsing geeft hij mee. Het is een zwabber bumper. De bumperswitch is met een 2-aderig snoer met een ingang verbonden, tussen in en Gnd.

Neutral.asm

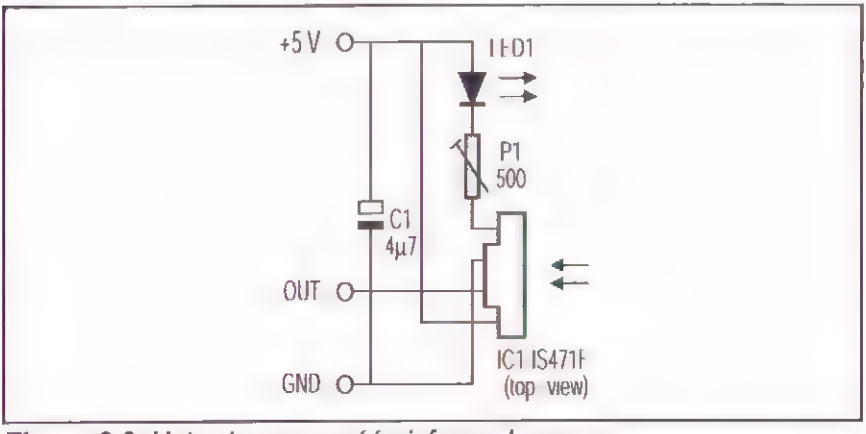
Met behulp van het programma Neutral.asm, voorgesteld in listing 6-1, kunnen de servo's neutraal worden gesteld. De bytes op adres 000C bepalen de draaisnelheid van servo 1 respectievelijk servo 2. Een waarde van 80 is de standaardwaarde (neutraal). Het instellen moet vrij nauwkeurig gebeuren.

De infrarood sensors

De infrarood sensors zijn gebaseerd op [2] en het schema ervan is getekend in figuur 6-2. De IS471F bevat een oscillator voor de sturing van de infrarood LED, waardoor de schakeling heel eenvoudig is.

```
; Listing 6.1: Neutral.asm, aid in adjusting servos
;
neutral   son
          p = neutral1      ; point to neutral string byte
          v0, v1 - mp       ; copy into v0, v1
          v0 to s1          ; copy v0 into servo 1
          v1 to s2          ; and v1 into servo 2
          break             ; return to command mode
neutral1  bytes 8080
```

Listing 6-1: *De listing van Neutral.asm.*



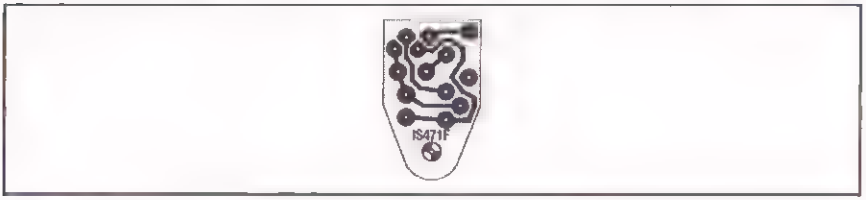
Figuur 6-2: *Het schema van één infrarood sensor.*

ONDERDELENLIJST

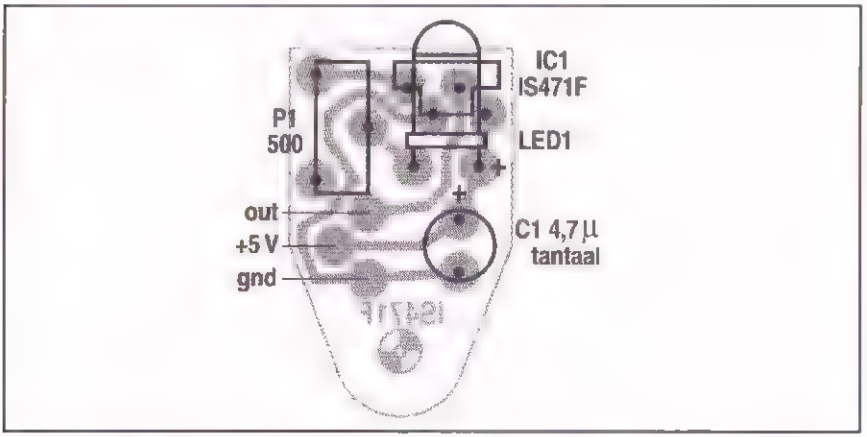
IC1	IS471F, Conrad best. nr. 185094
LED1	SFH409, Conrad best. nr. 183776
P1	Piher instelpotentiometer 500 Ω, 6 mm staand
C1	4,7 µF, 35 V tantaal, RM 2,54
1	print
1	3-aderig snoer
1	3-pens female SIL printheader

De bouw van de sensor

Figuur 6-3 laat de print zien en figuur 6-4 de opstelling van de onderdelen. Ook nu is de print niet op ware grootte getekend. Wij verwijzen u weer naar www.vego.nl/chip. Druk het TIF-bestand twee maal af op transparante folie met als afmetingen 11 x 18 mm.



Figuur 6-3: *De print van één infrarood sensor.*



Figuur 6-4: *De componentenopstelling van de print.*

De infrarood LED zit boven de sensor. Tussen de LED en de sensor zit een stukje zwart papier, anders ziet de sensor de LED. Aan het einde van het sensorsnoer is een driepolige female printheader gesoldeerd, met over de verbindingen krimpkousjes. Deze past op de ingangen van Chip. De printheader is met een figuurzaagje voor metaal van de strip afgezaagd. De sensors zijn niet zo gevoelig: zo'n 15 tot 20 cm, maar dat is hier juist voldoende. In figuur 6-5 (zie laatste pagina van dit hoofdstuk) is de montage van de sensoren op het chassis voorgesteld.

Het robotprogramma Robot.asm

Listing 6-2 is het volledige robotprogramma Robot.asm voor Chip. Na assemblage kan het in de EEPROM worden geladen, die is dan voor circa 20 % gevuld. Het programma wordt gestart door de schakelaar aan te zetten, terwijl de drukknop S1 is ingedrukt of de autostart jumper te plaatsen. De robot begint niet direct te rijden, eerst is er de count down van tien seconden. Dit is op het LCD te zien. Dan gaat de robot rijden.

De snelheid en richting worden door middel van toevalsgetallen uit een tabel gehaald. In eerste instantie werd de richting volkomen willekeurig gekozen, maar dat was niet leuk. Het is veel spannender als de robot enigszins zwalkend vooruit rijdt. Het lijkt dan, of hij tegen een obstakel aan zal botsen, maar meestal rijdt hij er net langs en soms botst hij er tegen op. Als hij er tegen op botst (bumper collision), gaat hij eerst een stukje achteruit en draait dan links- of rechtsom. Dit hangt weer af van een toevalsgetal, nadien begint hij weer vooruit te rijden.

Obstakels

Als een obstakel wordt gezien door een IR-sensor, dan gaat de robot een stukje achteruit, terwijl hij iets weg draait van het obstakel en dan iets vooruit, eveneens wegdraaiend van het obstakel, waarna hij weer zwalkend vooruit gaat. Als direct, nadat een IR-sensor een obstakel heeft gezien, de andere IR-sensor een obstakel ziet, dan wordt dit afgehandeld als een bumperbotsing. Terwijl de robot rond rijdt worden op het display mededelingen gezet over de stand van zaken. Als obstakels worden herkend piept de robot. In de listing is te zien, dat veelvuldig gebruik wordt gemaakt van toevalsgetallen en dat het merendeel van de te gebruiken gegevens in tabellen staat. Door middel van de pointer kunnen deze waarden snel in variabelen worden geladen.

Rijgedrag

Het is werkelijk geen vertoning zoals de Chip-robot door de kamer rijdt. Iedereen die het ziet is met stomheid geslagen. Hij rijdt weliswaar niet zo snel, maar het is de onvoorspelbaarheid van zijn rijden, dat het spannend maakt. Soms zit hij in een hoek (een verhuisdoos is heel leuk) en denk je: "daar komt t'ie nooit uit", en het volgende ogenblik rijdt hij alweer rustig zwalkend rond. Als hij vast zit, komt dat bijna altijd doordat de bumper komt klem te zitten, bijvoorbeeld tussen twee stoelpoten. Tegen dergelijke hinderlagen is Chip niet opgewassen.

```
; Listing 6.2: Robot.asm
; a robot program for Chip
;
; The robot mechanics are based on the robot from parallax.
; The infrared sensors are based on the IS471F integrated
; circuit.
;
; connections:              left servo = s1
;                           right servo = s2
```

```
;                          infra red detector left   in 0
;                          infra red detector right = in 1
;                          bumper switch - in 2
;
; note 1: detectors are normally high
; note 2: output f is set every second by the operating system
; note 3: do not use v0 for a random number, conflicts with
; 00nn instructions
; note 4: vb is used to get robot out of a corner, when there
; is an obstacle right or left, immediately followed by an
; obstacle on the other side, this is handled as a collision.
;
start     p - contxt       ; point p to count down text
          ld 0,f           ; and load display
          v0 = 0a          ; load v0 with count down delay
          v0 to sectimer   ; put into seconds timer
          set out e        ; set flag e for counting while
                           ; counting down
start1    skip out e = 1   ; skip jump to main while
                           ; flag e = set
          jp main
          skip out f - 0   ; skip call while flag f - 0
          call countdn     ; flag f = set by opsys, time to
                           ; call sub
          jp start1        ; and loop while flag e
;
; count down subroutine
;
countdn   res out f        ; reset calling flag f
          p = a stack      ; point p to a-stack for conversion
          v0 - sectimer    ; get seconds into v0
          v0 to 3dec mp    ; convert v0 to decimal on a stack
          p + 1            ; point to tens
          ld a,b           ; load display with tens and units
          skip v0 <> 00    ; skip resetting flag e while
                           ; seconds <> 00
          res out e        ; seconds = 00, reset flag e
          ret              ; return
;
main      son              ; enable servo drive
          p = wantxt       ; point p to wandering text
          rotate           ; rotate text on display
main1     v0 = timer       ; get the timer into v0
          skip v0 <> 00    ; skip jump to wander while v0
                           ; (timer) <> 00
          jp wander
main2     skip in 2 = 1    ; skip if no bumper collision
          jp collisi       ; yes, collision, jump to handling
                           ; routine
          call irsense     ; read infra red sensors
          skip va <> 00    ; if va <> 00, there was an obstacle
          jp main1         ; no obstacle, loop
```

```
          skip va <> 01
          jp obsleft        ; va = 01, obstacle left
          skip va <> 02
          jp obsrigt        ; va = 02, obstacle right
          jp obsfron        ; va must be 03, obstacle front
;
wander    v1 = rnd,1f       ; v1 = random 0 to 1f
          v1 + 10           ; v1 = random 10 to 2f
          v1 to timer       ; load timer with random number v1
          v1 = rnd,07       ; v1 = random 0 to 7
          v1 to v0          ; copy v1 into v0
          v1 + v0           ; v1 = random number * 2
          p = tablew        ; point p to start of direction table
          p + v1            ; add v1 for random table entry
          v0, v1 = mp       ; get table values into v0 to v1
          v0 to s1          ; load servo drives
          v1 to s2
          vb = 00           ; reset obstacle right/left v.v flag
          jp main2          ; and jump back
;
collisi   call stopser      ; stop servo's
          stop rotate       ; stop text rotate, clear display
          p = coltxt        ; and show collision text
          ld 0,f
collis1   p = esccoll       ; point p to start of collision table
          v1 = rnd, 08      ; v1 = a random number 0 or 8
          p + v1            ; p points to esccoll or esccolr
          call escape       ; execute the instructions
          vb = 00           ; reset obstacle right/left v.v flag
          jp main1          ; and jump back
obsleft   call stopser      ; stop servo's
          stop rotate       ; stop text rotate, clear display
          p = leftxt        ; and show obstacle left text
          ld 0,f
          skip vb <> 03     ; skip if .not. obstacle
                            ; right/left or v.v.
          jp collisi
          p = escleft       ; point p to escape left table
          call escape       ; execute the instructions
          jp main1          ; and jump back
obsrigt   call stopser      ; stop servo's
          stop rotate       ; stop text rotate, clear display
          p = rigtxt        ; and show obstacle right text
          ld 0,f
          skip vb <> 03     ; skip if .not. obstacle
                            ; right/left or v.v.
          jp collisi
          p = escrigt       ; point p to escape right table
          call escape       ; execute the instructions
          jp main1          ; and jump back
obsfron   call stopser      ; stop servo's
          stop rotate       ; stop text rotate, clear display
```

```
          p   frotxt        ; and show obstacle front text
          ld 0,f
          jp collis1        ; let collision handle this
;
; stop servo's subroutine
;
stopser   p = stop          ; point p to stop servo values
          v0, v1 = mp       ; load values into v0 to v1
          v0 to s1          ; load the servo's
          v1 to s2
          v0 = 06
          v0 to tone        ; give a short beep
          ret
;
; infra red sensors subroutine
;
irsense   va = 00           ; preset result value to 00
          v0 = 01           ; v0 is .or. mask left
          v1 = 02           ; v1 is .or. mask right
          skip in 0 = 1     ; skip no obstacle left
          va or v0          ; obstacle, .or into result
          skip in 1 = 1     ; skip no obstacle right
          va or v1          ; obstacle, .or. into result
          skip in 0 = 1     ; skip no obstacle left
          va or v0          ; obstacle, .or. into result
          vb or va          ; save result in vb
          ret
;
; escape subroutine
;
escape    v0, v5 = mp       ; load variables v0 to v5 from mp
          v0 to s1          ; v0 goes to servo 1
          v1 to s2          ; v1 goes to servo 2
          v2 to timer       ; v2 goes to the timer
escape1   v2 = timer        ; wait till timer = 00
          skip v2 = 00
          jp escape1
          v3 to s1          ; v3 goes to servo 1
          v4 to s2          ; v4 goes to servo 2
          v5 to timer       ; v5 goes to the timer
escape2   v5 = timer        ; wait till timer = 00
          skip v5 = 00
          jp escape2
          call stopser      ; stop the servo's
          p = wantxt        ; point p to wantxt
          rotate            ; load display
          p = tablew        ; set course straight forward
          v0, v1 = mp
          v2 = 10           ; for appr. 0.6 seconds
          v2 to timer       ; necessary to detect left obstacle
          v0 to s1          ; immediately after right obstacle
                            ; or vice versa
```

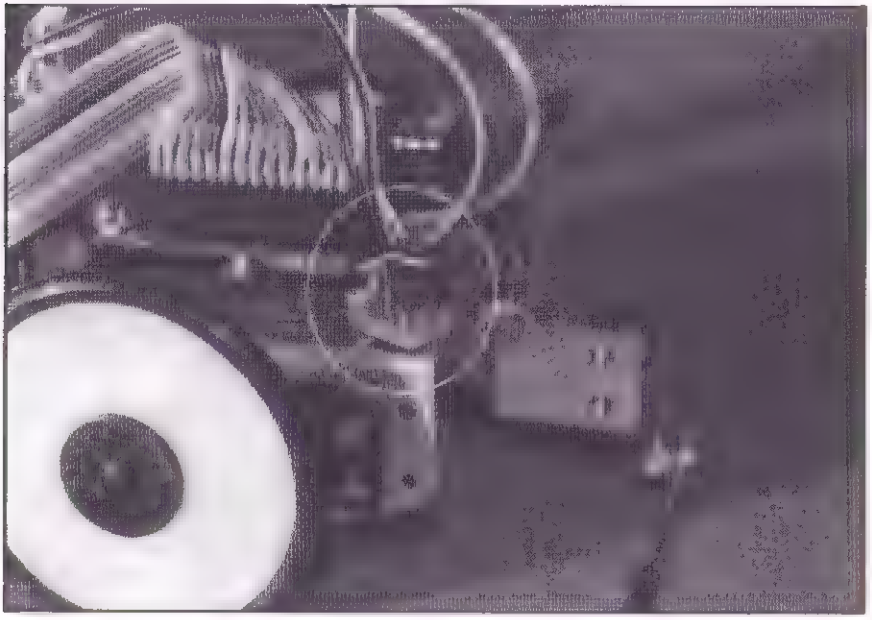
```
          v1 to s2          ; load servo drives
          ret               ; and return
;
; the random direction table for wandering
; note the duration is also set at random
;
tablew    bytes 778a        ; straight forward
          bytes 768a        ; increase speed left wheel
          bytes 758a        ; increase speed left wheel
          bytes 748a        ; increase speed left wheel
          bytes 778a        ; straight forward
          bytes 778b        ; increase speed right wheel
          bytes 778c        ; increase speed right wheel
          bytes 778d        ; increase speed right wheel
;
stop      bytes 8080
;
contxt    asciz "countdown    sec"
wantxt    asciz "Inspecting premises"
coltxt    asciz "Bumper Collision"
leftxt    asciz "Obstacle left!  "
rigtxt    asciz "Obstacle right! "
frotxt    asciz "Obstacle front! "
;
; the table for collision and obstacles
; note: esccoll and esccolr are selected at random
;
esccoll   bytes 847c20888835  ; go back, rotate left
          bytes 0000          ; fillers to get collision table
                              ; offset of 8
esccolr   bytes 847c20797930  ; go back, rotate right
escleft   bytes 80790f797910  ; go right back, turn left
                              ; a little
escrigt   bytes 87800f878710  ; go left back, turn right
                              ; a little
;
; end of robot
```

Listing 6-2: *De listing van Robot.asm.*

Referenties

[1] Elektuur, oktober en november 1999, www.stampsinclass.com, BoE documentation, robotcourse, BoE-Bot drawings in *.dxf en *.dwg format, www.antratek.nl, dutch distributor of parallax

[2] Elektuur mei 2000, Lego Robotics Invention System



Figuur 6-5: *De montage van een sensor op het chassis van de robot.*

7 Chip als klok

Inleiding

Als je met plezier aan het werk bent, vliegt de tijd voorbij. Een klok waarop je kunt zien of het tijd voor koffie is, is onmisbaar. Chip kan heel goed als klok worden gebruikt, maar om hem alleen daarvoor te gebruiken is onterecht, daarvoor is hij te knap. Omdat Chip weet welke week en welke dag het is, kan hij ook als verjaardagskalender worden gebruikt en om de puntjes op de i te zetten laten we hem tikken en geven hem een slagwerk waarbij, voor het slaan van de tijd, een melodietje is te horen.

Een klok

De realtime klok van Chip wordt gestuurd door de interrupt routine van timer 1. De klok werkt dus op de achtergrond en zo lang we Chip niet uitschakelen blijft hij precies op tijd lopen, mits het afregelbyte correct is ingesteld. De klok houdt de weken, dagen, uren, minuten en seconden (zie tabel 7-1) bij en is dus geschikt voor homesystemen omdat het dagnummer aangeeft of het een doordeweekse of een weekend dag is.

AAh	weken
ABh	dagen
ACh	uren
ADh	minuten
AEh	seconden

Tabel 7-1: *Chip's inwendige tijdregisters.*

Clock.asm

Clock.asm, zie listing 7-1, laat de subroutine zien die de tijd op het display zet. Het aardige aan deze routine is dat gebruik wordt gemaakt van de variabelen v0 en v1 om de posities op het display te adresseren. Hierdoor kunnen we een lus maken en zijn de conversie- en display-instructie slechts eenmaal nodig. Een nadeel is dat de routine wat meer tijd in beslag neemt. De klok wordt eenmaal per seconde op het display gezet, vaker is immers niet nodig. Het spaart processortijd en maakt het display rustiger om te zien. De klok kan worden ingesteld met het commando **time**, waarbij het aan de gebruiker wordt overgelaten om voor de zondag, dag zeven of dag één te kiezen.

```
; Listing 7.1: Clock.asm
; Clock, subroutine to show the week, day of week and time on
; the LCD.
; Note that we use variables to load the display,
; so we can use a loop.
;
main      p= clock2          ; point to the initial clock text
          ld 0,f             ; and load the initial display
main0     skip out f = 0     ; skip if the seconds flag is not set
          call clock         ; call the clock sub
          jp main0           ; loop
;
          org 20             ; let the clocksubroutine begin here
clock     res out f          ; reset the seconds flag
          v0   01            ; initialise v0 and v1
          v1 = 02
clock1    skip v0 <> 01      ; get the corresponding byte
          v3 = weeks         ; and load into v3
          skip v0 <> 04
          v3 = days
          skip v0 <> 07
          v3 = hours
          skip v0 <> 0a
          v3 = minutes
          skip v0 <> 0d
          v3 = seconds
          p = a-stack        ; we let p point to a-stack
          v3 to 3dec         ; and convert v3 to 3 decimals
          p+1                ; we don't need the hundreds
          ld v0,v1           ; show tens and units on display
          v0 + 03            ; let v0 and v1 point
          v1 + 03            ; to next display position
          skip v0 = 10       ; if v0 = 10 we are ready
          jp clock1          ; else we loop
          ret
clock2    asciz "   :  :  :  :   "
; this is the inititial display
```

Listing 7-1: *De listing van Clock.asm.*

Chip leert klok kijken

Chip moet ook kunnen klok kijken om te weten of er iets moet gebeuren. Met het programma Dattime.asm in listing 7-2 kan Chip klok kijken. De gebeur-tijd staat in de bytesstring **datstr1**, in dit geval week 41, dag 06, 09 uur, 15 minuten en 00 seconden precies. In de hoofdroutine wordt de pointer op de string gezet en de subroutine wordt aangeroepen. Als de klok gelijk is aan de stringtijd wordt vf <> 00 gemaakt waarop de roepende routine actie kan ondernemen. In het voorbeeld wordt een geluidssignaal

gegeven. De **dattime** subroutine laadt eerst de stringbytes in variabelen waarbij bit 7 van de weken wordt gestript. Vervolgens worden de variabelen vergeleken met de klok, tenminste als ze ongelijk 7Fh zijn. Als alles gelijk is, en in de tijdstring bit 7 van de weken 0 is, wordt vf gezet (<> 00) en het genoemde bit wordt gezet. Zodra de tijden ongelijk zijn geworden, wordt dit bit weer gestript. Een 7Fh byte is een joker, een doet er niet toe byte. Als we deze waarde als week invullen, wordt de string iedere week geldig en als we voor weken en dagen 7Fh invullen, iedere dag. Als we voor de seconden 7Fh invullen, hebben we een minuut de tijd voor de string ongeldig wordt. Voor de test op een weekend gebruiken we natuurlijk de instructie **vx = days** en testen vx. Er kunnen meerdere tijdstrings in een programma worden opgenomen en de string waarop de pointer staat wordt getest. In het voorbeeld is de verjaardag van auteur dezes opgenomen. De instructies **set out 4** en **res out 4** zijn opgenomen om met de scoop de tijden de kunnen meten.

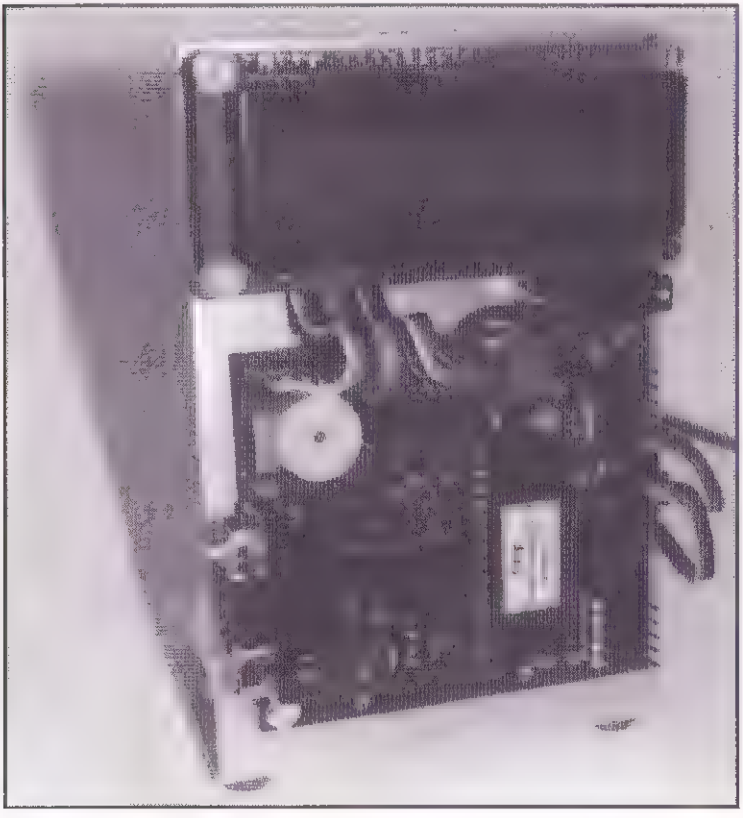
```
; Listing 7.2: Dattime.asm
;
main      p = datstr1
main1     set out 4
          call dattime
          res out 4
          skip vf = 00
          jp sound
          jp main1
sound     v0 - 80
          v0 to tone
          jp main1
;
datstr1   bytes 2906       ; weeks, days
          bytes 090f       ; hours, minutes
          bytes 00         ; seconds
;
; dattime, subroutine to compare the running clock against a
; date-time string
; pointed to by the pointer:
; p -> weeks, days, hours, minutes, seconds (all values hex!)
; when a dattim string byte is 7f, it will not be tested
; when the condition is met for the first time, vf will be set
; as flag (<> 00)
; and bit 7 of weeks, date time string will be set. As soon as
; the condition
; is no more valid this bit wil be reset.
;
dattime   v0,v4 - mp       ; move string date and time into
                           ; v0...v4
```

```
          vf = 7f          ; strip bit 7 from v0
          v0 and vf
          skip v0 <> 7f    ; skip if weeks <> 7f
          jp dattim1       ; weeks = 7f, so do not test
          vf = weeks       ; get clock weeks
          skip v0 = vf     ; skip if equal
          jp dattim7       ; not equal, jump
dattim1   skip v1 <> 7f    ; skip if days <> 7f
          jp dattim2       ; days = 7f, so do not test
          vf = days        ; get clock days
          skip v1 = vf     ; skip if equel
          jp dattim7       ; not equal, jump
dattim2   skip v2 <> 7f    ; skip if hours <> 7f
          jp dattim3       ; hours = 7f, so do not test
          vf = hours       ; get clock hours
          skip v2 = vf     ; skip if equal
          jp dattim7       ; not equal, jump
dattim3   skip v3 <> 7f    ; skip if minutes <> 7f
          jp dattim4       ; minutes = 7f, so do not test
          vf = minutes     ; get clock minutes
          skip v3 = vf     ; skip if equal
          jp dattim7       ; not equal, jump
dattim4   skip v4 <> 7f    ; skip if seconds <> 7f
          jp dattim5       ; seconds = 7f, so do not test
          vf = seconds     ; get clock seconds
          skip v4 = vf     ; skip if equal
          jp dattim7       ; not equal, jump
;
; dattim string is equal to the clock
; when this happens for the first time, bit 7 of v0 (weeks)
; will be 0,
; it must be set to 1 and vf must be made <> 00
;
dattim5   v0, v0 = mp
          vf = 80          ; vf is bitmask
          vf and v0        ; strip d6...d0
          skip vf = 00     ; skip if d7 = 0
          jp dattim6       ; jump, this dattim skip has already
                           ; been taken
          vf = 80          ; vf is .or. bit 7
          v0 or vf         ; set 7 of v0
          v0, v0 to mp     ; write back to dattim string
          skip a           ; skip always, let vf remain 80
dattim6   vf = 00          ; entry point for skip already taken,
                           ; reset flag
          ret
dattim7   v0, v0 to mp     ; write back to dattim string
          vf = 00          ; reset flag vf
          ret
```

Listing 7-2: *De listing van Dattime.asm.*

Een hi-tech klok

Onze wekker was kapot. We waren eraan gehecht want ruim 35 jaar heeft hij trouw voor ons getikt. Het was een van de eerste wekkers waarbij de onrust via een transistor elektromagnetisch werd aangedreven. De rest was zuiver mechanisch. Onze gedachten gingen naar Chip. Deze bevat immers alle componenten voor een aardig klokje en ook voor wat betreft de software waren er interessante mogelijkheden.
Van een plaatje aluminium 160 mm hoog, 82 mm breed, dik 1,5 mm hebben we aan de onderkant een strook van 40 mm omgezet, iets meer dan 90 graden, zie figuur 7-1.



Figuur 7-1: *Het high tech klokje waarschuwt bij verjaardagen.*

De omgezette strook is de bodem. Boven aan de voorkant hebben we het display gemonteerd, met daaronder Chip. Tussen het display en Chip zit

de aan/uit-schakelaar. Weliswaar gebruikt Chip slechts 8 mA, maar dat is teveel voor een klokje met batterijvoeding. Daarom is een (ongestabiliseerd) netstekervoedinkje gebruikt dat 300 mA kan leveren en dat omschakelbaar is tussen 3 V, 4,5 V, 6 V, 7,5 V, 9 V en 12 V. Voor de voeding van Chip is een low drop 5 V stabilisator gebruikt. Omdat het makkelijk is als het klokje ook in het donker afleesbaar is, hebben we een display met backlight gebruikt dat via een serieweerstand van 27 Ω, 2 W op de voeding is aangesloten. Met de spanningkeuze kan het backlight van weinig tot heel helder worden ingesteld.

Kroonstrip voor de aansluitingen

Aan de achterkant van de constructie hebben wij een kroonstrip gemonteerd waarop alle externe bedrading kan worden aangesloten, zie figuur 7-2.

Figuur 7-2: *Handig, de kroonsteenstrip aan de achterkant voor montage van onderdelen.*

Opmerking

Bij 12 V wordt de weerstand vrij warm, hij dissipeert dan ongeveer 1,5 W.

Tikken en melodietje

Onze wekker tikte, dus het nieuwe klokje moest ook tikken. Net als een echte klok moest hij ook bij de hele uren slaan en om het geluidenpallet compleet te maken moest hij, voor dat slaan van de uren, een muziekje spelen.

Chip onthoudt verjaardagen

Toch ontbrak er nog iets. Weliswaar hebben we een verjaardagskalender, maar we kijken daar te weinig op en vergeten verjaardagen. Het klokje kon daarbij helpen, want Chip kan immers klok kijken.
Zo hebben we nu een heel bijzonder klokje, dat voor een technisch iemand een plezier is om naar te kijken, maar dat ook wat betreft functionaliteit zijn weerga niet kent.
Om de muziek te laten horen moet op **servo 1** een piëzo sounder worden aangesloten, eenzelfde type als op de Chip print zit. En om een roterende felicitatiewens te kunnen stoppen, moet op **input 0** (en Gnd) een drukknop worden aangesloten.

Calclock.asm

Het klokprogramma Calclock.asm (Calendar and clock) is te zien in listing 7-3. Eerst wordt de PWM-timer omgeprogrammeerd om muziek te spelen. In de hoofdlus **mainlo(op)** worden vlag 5 en vlag 6 getest. Als vlag 5 is gezet wordt een muziekje (**Chimes**) gespeeld en als vlag 6 is gezet wordt het aantal uren geslagen (**Hourwrk**). Er zijn vier liedjes, via een toevalsgetal (random number) wordt er een gekozen. Omdat de Chip 24 uren aangeeft, wordt vanaf 13:00 uur een correctie aangebracht en bij 00:00 uur (12 uur 's nachts) wordt twaalf maal geslagen. Eenmaal per seconde wordt de tijd getoond door subroutine **Clock**. De subroutines **Clock**, **Chimes** en **Hourwrk** sturen de vlaggen. Aardig is dat bij het slaan van de uren de clock doorloopt.
De verjaardagen worden niet alle iedere seconde getest, maar per seconde één, te beginnen bij seconde 45. Eerst wordt de pointer op **datsent** gezet, dan wordt de pointer op een (bij een seconde horende) **dattim** string gezet. Als de pointer niet is veranderd, is dat de zien aan de byte 00 waarop hij wijst en gaan we terug naar **mainloop**. Als hij wel op een dattim string staat, zal subroutine **dattime** vf <> 00 maken bij een verjaardag op de ingestelde tijd. Precies zes 6 posities lager staat de felicitatieboodschap die gaat roteren op het display. Het roteren stopt als input 0 wordt laag gemaakt of als het 10:00 uur wordt.

Practisch gezien worden alle tijdstrings eenmaal per minuut getest. Daarom zijn in de strings de seconden op don't care (7fh) ingesteld. Anders zouden we veel boze gezichten krijgen. Chip werkt met weken en dagen, daarom moeten de verjaardagen worden omgezet in deze notatie en natuurlijk in hex. Aan het begin van ieder jaar moeten de voor dat jaar geldende strings worden geladen, maar dat betreft dan alleen het deel beginnende met **org 400**.
In de dattim-strings staat een tijd van 07:45 uur. Dat is de tijd waarop bij een verjaardag de gelukwens verschijnt. Misschien heeft u liever dat dit op een heel uur gebeurt. Dat kan natuurlijk, vandaar ook dat de tests pas beginnen vanaf seconde 45, dan is het slagwerk al lang afgelopen.

```
; Listing 7.3: Calclock.asm (calendar and clock)
; lcd clock plays melody on the hour and chimes hours
; congratulates family and friends on their birthday
;
; artimer register adressen and port b data register
;
arsc1     equ 8d7          ; ar status control register 1
armc      equ 8d5          ; ar mode control register
arrc      equ 8d9          ; ar reload register
arcp      equ 8da          ; ar compare register
drb       equ 8c1          ; port b data register
;
init0     p = clocdis      ; point to the initial clock text
          ld 0,f           ; and load the initial display
          p = initstr      ; point to string with intialize
                           ; values and..
          v6,va = mp       ; load initialize values into v6...va
          p = drb          ; point to drb and..
          v6,v6 to mp      ; connect servo 1 to pb7 and reset
                           ; servo 1
          p = armc         ; point to artimer mode control
                           ; register and..
          v7,v7 to mp      ; set artimer = off and pwm = off
          p= arcp          ; point to artimer compare register
                           ; and..
          v8,v8 to mp      ; load with f0
;
; v7 = 80,                 ; to set artimer = off and pwm = off
; v9   e0,                 ; to set artimer = on and pwm   on
; va = 04,                 ; to increment pointer to next music
                           ; table entry
;
mainlo0   skip out 5 = 0   ; skip if chimes flag is not set
          jp chimes        ; jump to chimes
          skip out 6 = 0   ; skip if hours flag is not set
          jp hourwrk       ; jump to hours
```

```
mainlo1   skip out f = 1   ; skip if the seconds flag is set
          jp mainlo0
          res out f
          call clock       ; call the clock sub
;
; Test for seconds: 2d...36 (decimal: 45...54), set
; corresponding pointer
;
          v0 = seconds
          p = datsend      ; set pointer to 00 byte
          skip v0 <> 2d    ; from now on, set every second
          p = datstr1      ; the pointer on a dattime string
          skip v0 <> 2e    ; so the tests will be distributed
          p = datstr2      ; in time
          skip v0 <> 2f
          p = datstr3
          skip v0 <> 30
          p = datstr4
          skip v0 <> 31
          p = datstr5
          skip v0 <> 32
          p = datstr6
          skip v0 <> 33
          p = datstr7
          skip v0 <> 34
          p = datstr8
          skip v0 <> 35
          p = datstr9
          skip v0 <> 36
          p = datstrA
;
          v0,v0 = mp       ; get first string byte
          skip v0 <> 00
          jp mainlo0       ; if byte = 00, pointer not set jump
back
          call dattime     ; compare clock and string
          skip vf <> 00    ; skip if equal
          jp mainlo0       ; not equal, jump
          v0 = 06          ; set pointer to message
          p + v0
          rotate           ; rotate message on display,
                           ; give beebs
mainlo2   v0 = hours
          skip v0 <> 0a    ; rotate text till 10 am o'clock...
          jp mainlo3
          skip in 0 = 0    ; ...or till button in 0 is pressed
          jp mainlo2
mainlo3   stop rotate
          p = clocdis      ; point to the initial clock text
          ld 0,f           ; and load the initial display
          jp mainlo0
;
```

```
hourwrk   vb = rnd, 03      ; vb is random 0...3
          skip vb <> 00     ; set p acording random to melody
          p = melody
          skip vb <> 01
          p = hymne
          skip vb <> 02
          p = stars
          skip vb <> 03
          p = auclair
          call musibox      ; play melody
          v8 = hours        ; get hours into v8
          skip v8 <> 00
          v8 - 0c           ; if hours is null, make hours 12
          vb   0c
          skip v8 > vb      ; double skip to invert skip
                            ; condition
          skip a
          v8   vb           ; so this is executed if skip not
                            ; true
          vb   30
          vb to timer       ; set timer for delay between melody
                            ; and chimes
          res out 6         ; reset hours flag, to not get here
                            ; again
          set out 5         ; set chimes flag to chime hours
          jp mainlo1
;
chimes    skip v8 <> 00
          res out 5         ; reset chimes flag if all hours are
                            ; chimed
          vb = timer
          skip vb = 00
          jp mainlo1        ; jump back if timer not yet zero
          p = beet          ; point to chime sound
          call musibox      ; chime
          v8 + ff           ; hours = hours - 1
          vb = 20
          vb to timer       ; set timer for delay between chimes
          jp mainlo1
;
clocdis   asciz "   :  :  :  :   "
; this is the inititial display
initstr   bytes c080f0e004 ; presets for v6...va
;
; Clock, subroutine to show the week, day of week and time on
; the LCD.
; Note that we use variables to load the display,
; so we can use a loop.
;
clock     v0 = 01           ; initialise v0 and v1
          v1 = 02
          v0 to tone        ; tick every second
```

```
clock1    skip v0 <> 01     ; get the corresponding byte
          v3 - weeks        ; and load into v3
          skip v0 <> 04
          v3 - days
          skip v0 <> 07
          v3   hours
          skip v0 <> 0a
          v3   minutes
          skip v0 <> 0d
          v3   seconds
          p = a-stack       ; we let p point to a-stack
          v3 to 3dec        ; and convert v3 to 3 decimals
          p+1               ; we don't need the hundreds
          ld v0,v1          ; show tens and units on display
          v0 + 03           ; let v0 and v1 point
          v1 + 03           ; to next display position
          skip v0 = 10      ; if v0 = 10 we are ready
          jp clock1         ; else we loop
          skip v3 = 00
          ret               ; return if v3 <> 00
          v3 - minutes
          skip v3 <> 00
          set out 6         ; set out 6 if minutes = 00 (and
                            ; seconds = 00)
          ret
;
; dattime, subroutine to compare the running clock against a
; date-time string
; pointed to by the pointer:
; p -> weeks, days, hours, minutes, seconds (all values hex!)
; when a dattim string byte is 7f, it will not be tested
; when the condition is met for the first time, vf will be
; set as flag (<> 00)
; and bit 7 of weeks, date-time string will be set. As soon
; as the condition
; is no more valid this bit wil be reset.
;
dattime   v0,v4 = mp        ; move string date and time int
                            ; o v0...v4
          vf = 7f           ; strip bit 7 from v0
          v0 and vf
          skip v0 <> 7f     ; skip if weeks <> 7f
          jp dattim1        ; weeks = 7f, so do not test
          vf = weeks        ; get clock weeks
          skip v0 = vf      ; skip if equal
          jp dattim7        ; not equal, jump
dattim1   skip v1 <> 7f     ; skip if days <> 7f
          jp dattim2        ; days = 7f, so do not test
          vf = days         ; get clock days
          skip v1 = vf      ; skip if equel
          jp dattim7        ; not equal, jump
dattim2   skip v2 <> 7f     ; skip if hours <> 7f
```

```
            jp dattim3          ; hours = 7f, so do not test
            vf = hours          ; get clock hours
            skip v2 = vf        ; skip if equal
            jp dattim7          ; not equal, jump
dattim3     skip v3 <> 7f       ; skip if minutes <> 7f
            jp dattim4          ; minutes = 7f, so do not test
            vf = minutes        ; get clock minutes
            skip v3 = vf        ; skip if equal
            jp dattim7          ; not equal, jump
dattim4     skip v4 <> 7f       ; skip if seconds <> 7f
            jp dattim5          ; seconds = 7f, so do not test
            vf = seconds        ; get clock seconds
            skip v4 = vf        ; skip if equal
            jp dattim7          ; not equal, jump
;
; dattim string is equal to the clock
; when this happens for the first time, bit 7 of v0 (weeks)
; will be 0, it must be set to 1 and vf must be made <> 00
;
dattim5     v0, v0 = mp
            vf = 80             ; vf is bitmask
            vf and v0           ; strip d6...d0
            skip vf = 00        ; skip if d7 = 0
            jp dattim6          ; jump, this dattim skip has already
                                ; been taken
            vf = 80             ; vf is .or. bit 7
            v0 or vf            ; set 7 of v0
            v0, v0 to mp        ; write back to dattim string
            skip a              ; skip always, let vf remain 80
dattim6     vf = 00             ; entry point for skip already taken,
                                ; res flag
            ret
dattim7     v0, v0 to mp        ; write back to dattim string
            vf = 00             ; reset flag vf
            ret
;
; musibox
; plays music from table, using artimer (pwm-timer)
; before entry pointer must be set on table start
;
musibox     save p              ; save pointer, will be needed later
musibo0     rest p              ; point into music table
            vb,ve = mp          ; vb=note, vc=octave, vd=duration,
                                ; ve=delay
            p + va              ; point to next music table entry
            save p              ; save pointer
            skip vb <> 00       ; skip if note <> 00
            ret                 ; ret if note = 00
            p = arrc            ; point to ar reload register..
            vb,vb to mp         ; load note
            p = arscl           ; point to ar status control register
                                ; 1 and..
```

```
            vc,vc to mp       ; ...load octave into predivider
            vd to timer       ; load duration into timer
            p = armc          ; point to artimer mode control
                              ; register and..
            v9,v9 to mp       ; set artimer = on and pwm = on
musibo1     vd = timer        ; wait for duration of note
            skip vd - 00
            jp musibo1
            v7,v7 to mp       ; set artimer = off and pwm = off
musibo2     ve + ff           ; wait for delay time between notes
            skip ve = 00
            jp musibo1
            jp musibo0
;
melody      bytes 88a10407
            bytes 81a10407
            bytes 88a10407
            bytes 71811007
            bytes 00
;
hymne       bytes 81a10807
            bytes 81a10807
            bytes 88a10807
            bytes 95a10807
            bytes 95a10807
            bytes 88a10807
            bytes 81a10807
            bytes 71a10807
            bytes 60a10807
            bytes 60a10807
            bytes 71a10807
            bytes 81a10807
            bytes 81a11007
            bytes 71a11007
            bytes 81a10807
            bytes 81a10807
            bytes 88a10807
            bytes 95a10807
            bytes 95a10807
            bytes 88a10807
            bytes 81a10807
            bytes 71a10807
            bytes 60a10807
            bytes 60a10807
            bytes 71a10807
            bytes 81a10807
            bytes 71a11007
            bytes 60a11007
            bytes 00
;
stars       bytes 60a11007
            bytes 95a11007
```

```
          bytes 88a10407
          bytes 81a10407
          bytes 71a10407
          bytes 60811007
          bytes 95a11007
          bytes 88a10407
          bytes 81a10407
          bytes 71a10407
          bytes 60811007
          bytes 95a11007
          bytes 88a10407
          bytes 81a10407
          bytes 88a10407
          bytes 71a11007
          bytes 00
;
auclair   bytes 60a10807
          bytes 60a10807
          bytes 60a10807
          bytes 71a10807
          bytes 81a11007
          bytes 71a11007
          bytes 60a10807
          bytes 81a10807
          bytes 71a10807
          bytes 71a10807
          bytes 60a11007
          bytes 00
;
beet      bytes 88810407
          bytes 88810407
          bytes 71811007
          bytes 00
;
          org 400              ; birthdate strings and
                               ; congratulations
;
datstr1 bytes 0206             ; weeks = 02d, days = 06d
          bytes 072d           ; hours = 07d, minutes = 45d
          bytes 7f00           ; seconds = don't care, filler
messag1   asciz "Hoera, Willem is jarig!"
;
datstr2 bytes 1102             ; weeks = 17d, days = 02d
          bytes 072d           ; hours = 07d, minutes = 45d
          bytes 7f00           ; seconds = don't care, filler
messag2   asciz "Hoera, Marjan is jarig!"
;
datstr3 bytes 1405             ; weeks = 20d, days = 05d
          bytes 072d           ; hours = 07d, minutes = 45d
          bytes 7f00           ; seconds = don't care, filler
messag3   asciz "Ons Jan Peter is jarig!"
;
```

```
datstr4 bytes 1c03          ; weeks = 28d, days = 03d
        bytes 072d          ; hours = 07d, minutes = 45d
        bytes 7f00          ; seconds = don't care, filler
messag4 asciz "Remember onze Reinier! "
;
datstr5 bytes 2105          ; weeks = 33d, days = 05d
        bytes 072d          ; hours = 07d, minutes = 45d
        bytes 7f00          ; seconds = don't care, filler
messag5 asciz "Hoera, Sara is jarig!  "
;
datstr6 bytes 2903          ; weeks = 41d, days = 03d
        bytes 072d          ; hours = 07d, minutes = 45d
        bytes 7f00          ; seconds = don't care, filler
messag6 asciz "Hoera, Wil is jarig!   "
;
datstr7 bytes 2a05          ; weeks = 42d, days = 05d
        bytes 072d          ; hours = 07d, minutes = 45d
        bytes 7f00          ; seconds = don't care, filler
messag7 asciz "Hoera, Esther is jarig!"
;
datstr8 bytes 2b04          ; weeks = 43d, days = 04d
        bytes 072d          ; hours = 07d, minutes = 45d
        bytes 7f00          ; seconds = don't care, filler
messag8 asciz "Hoera, Bobsie is jarig!"
;
datstr9 bytes 2e02          ; weeks = 46d, days = 02d
        bytes 072d          ; hours = 07d, minutes = 45d
        bytes 7f00          ; seconds = don't care, filler
messag9 asciz "Tante Andrea is jarig! "
;
datstrA bytes 3203          ; weeks = 50d, days = 03d
        bytes 072d          ; hours = 07d, minutes = 45d
        bytes 7f00          ; seconds = don't care, filler
messagA asciz "Hoi, Michael is jarig! "
datsend bytes 00            ; zero byte to check for unchanged
                            ; pointer
```

***Listing 7-3:** De listing van Calclock.asm.*

Tot slot

In de secondentests staan heel wat skips, waardoor dit programmadeel wel heel duidelijk is, maar een rechtgeaarde programmeur vraagt zich af of dit niet korter kan. Dat kan vrij eenvoudig met twee optellingen en twee tests van de carry (vf). De verkregen waarde is de index (0...) in een tabel en moet worden vermenigvuldigd met de offset tussen de entrees van de tabel. Alle felicitatiewensen moeten dan wel even lang zijn, wat met de skipmethode niet het geval is.
Nuttig om te weten is ook dat instructie **p + vx** een 16 bit optelling is.

8 Chip als Homesystem

Inleiding

We gaan onze woning automatiseren met Chip. Belangrijke factoren daarbij zijn de tijd en de temperatuur zowel binnen als buiten, onder andere om de kachel aan of uit te kunnen zetten afhankelijk van temperatuur en tijd. De tijd heeft alles te maken met de klok van Chip en daar kunnen we, dankzij hoofdstuk 7, mee lezen en schrijven. Ook willen we graag weten of het al donker genoeg is om de lamp aan de doen of dat de lamp juist uit kan. Chip Homesystem combineert al deze zaken (en meer) in een juist verband en maakt ons leven iets makkelijker. Bovendien wordt geld bespaard omdat de verwarming niet onnodig aanstaat.

Elektronische temperatuursensor

De temperatuursensor is gebaseerd op de KTY10-6. Dat is een PTC met een nominale weerstand (±1 %) van 2 kΩ bij 25 °C. In tabel 8-1 staan de weerstandswaarden voor enkele temperaturen. Let trouwens wel op de 6 in het typenummer want een ander getal duidt op een andere nominale weerstandswaarde.

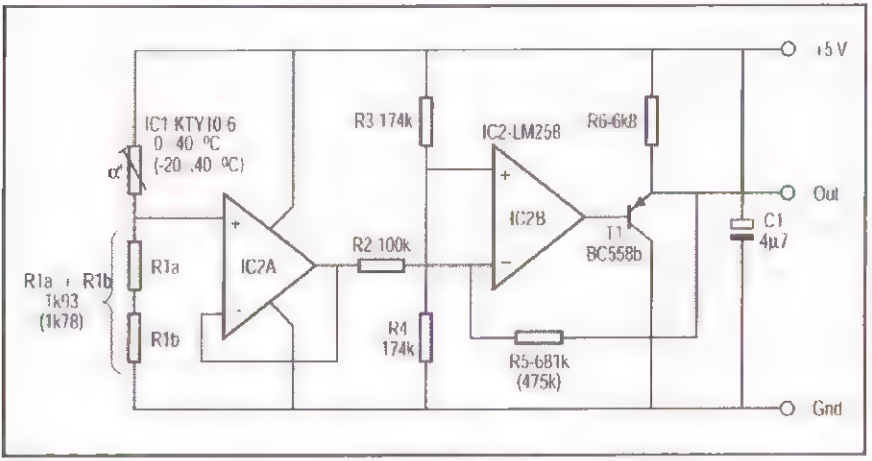
Temp	KTY10-6
°C	Ω
-20	1.387
-10	1.513
0	1.645
10	1.783
20	1.926
30	2.075
40	2.230

***Tabel 8-1:** Weerstand van KTY10-6 sensor bij verschillende temperaturen.*

Het schema

Het schema van de temperatuurmeter is getekend in figuur 8-1. Er zijn twee sensors noodzakelijk, een voor het buiten temperatuurbereik van -20 tot +40°°C en een voor het binnen temperatuurbereik van 0 tot +40 °C. De sensors zijn identiek op twee weerstandswaarden na, tussen de haakjes zijn die voor de buitensensor opgenomen. In wezen is de sensor in een

brugschakeling opgenomen, waarbij het spanningsverschil in de brug wordt versterkt. Het IC (LM258) heeft als uitgangstrap een NPN emittervolger. Om het spanningsbereik van de uitgang groter te maken, is T1 opgenomen.



Figuur 8-1: *Het schema van de temperatuursensor.*

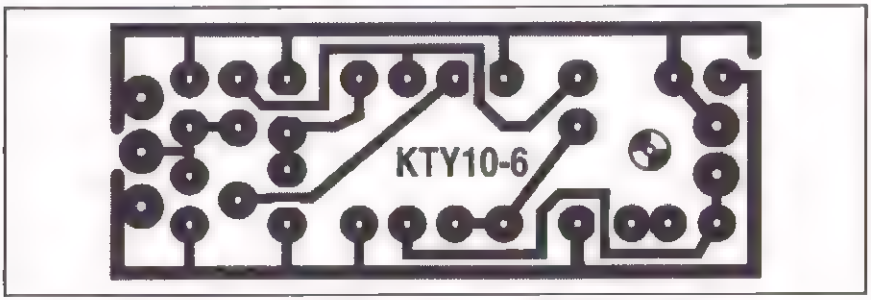
ONDERDELENLIJST

R1	1,93 kΩ
R1'	1,78 kΩ
R2	100 kΩ
R3	174 kΩ
R4	174 kΩ
R5	681 kΩ
R5'	475 kΩ
R6	6,8 kΩ
C1	4,7 µF, 16 V printelco
T1	BC558B
IC1	KTY10-6
IC2	LM258

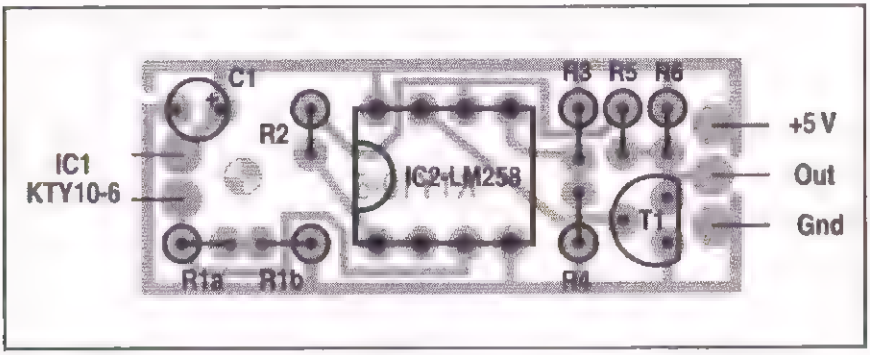
De "middentemperatuur" voor de binnensensor is 20 °C en voor de buitensensor 10 °C. De brug is dan in evenwicht (zie ook tabel 8-1) en de uitgangsspanning is de halve voedingsspanning. De sensor is bij Chip vrij ongevoelig voor schommelingen in de voedingsspanning en, omdat slechts een deel van de temperatuur/weerstandcurve van de KTY10-6 wordt gebruikt, vrijwel lineair.

De bouw van de schakeling

Figuur 8-2 laat de print zien en figuur 8-3 de componentenopstelling. Ook nu is de print niet op schaal 1/1 voorgesteld. Ga naar www.vego.nl/chip en open de print in een beeldbewerkingsprogramma. Druk af op transparante folie met als afmetingen 34 x 13 mm^2. Gebruik bij voorkeur 1 % metaalfilm weerstanden, die zijn het minst temperatuurgevoelig.



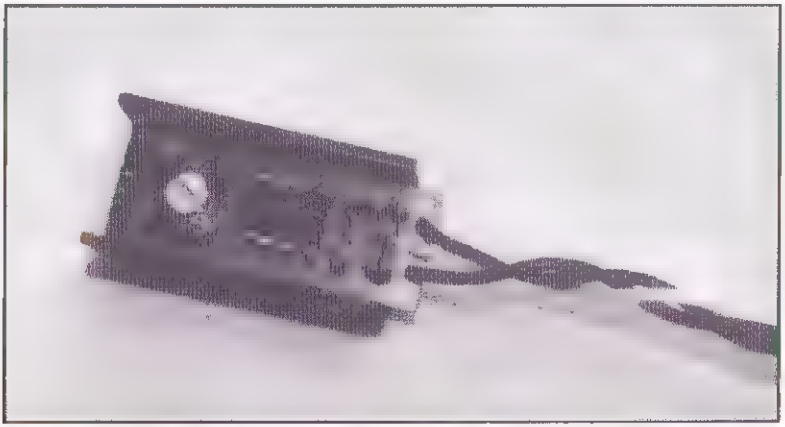
Figuur 8-2: *De print voor de temperatuursensor.*



Figuur 8-3: *De componentenopstelling van de temperatuursensor.*

De montage van de print

Het printje past in het U-vormig koelelement voor TO-220 halfgeleiders, zie figuur 8-4. De sensor wordt aan zijn draden omgebogen zodat hij vlak op de soldeerzijde ligt. Aan de aansluitkant komt als afstandstuk tussen de print en het koelelement een strookje epoxy en dan wordt het printje met een M3 boutje in het koelelement vastgezet, waarbij de sensor wordt vastgeklemd tussen de print en de koelvin. De lengte van het sensorsnoer is niet kritisch, bij onze opstelling was dat voor de buitensensor circa 5 meter.



Figuur 8-4: *De montage van de sensor en de print.*

Opmerking

Het is mogelijk voor de binnensensor een LM358 (0...+70 °C) te gebruiken. Dat kan ook voor de buitensensor, als men het printje binnenskamers houdt en de sensor via een snoertje verbindt. De LM258 heeft namelijk een groter temperatuurbereik (-25...+85 °C).

Meten van de binnentemperatuur

Listing 8-1 toont het programma Tempin.asm om de, door de binnensensor op input 1, gemeten waarde om te rekenen naar °C en op het display te zetten. Heel aardig is dat kan worden afgetrokken door op te tellen. Het schalen van de waarde is heel eenvoudig door Chip's 16 bit vermenigvuldig- en deelinstructies.

Meten van de buitentemperatuur

Het programma Tempout.asm voor de buitensensor (listing 8-2) is iets ingewikkelder omdat nu ook een negatieve temperatuur mogelijk is. Als dat het geval is, wordt een minteken op het display gezet en wordt de absolute waarde van de temperatuur genomen.

Nauwkeurigheid

Door de initiële nauwkeurigheid van de KTY10-6 en de 1 % metaalfilmweerstanden in de sensorschakeling zal de temperatuur vrij goed kloppen. Als dat niet het geval is, is het ijken van de sensor een interessant klusje voor een regenachtige zondag.

```
; Listing 8.1: Tempin.asm
;
main        skip out f = 0
            call tempin
            jp main
;
; tempin, subroutine to measure interior temperature with
; kty 10-6
; chip temp-in sensor connected to input 1
;
tempin      res out f           ; reset seconds flag
            p = tempin1         ; point to text
            ld 0,f              ; load display
            v0 = ana 1          ; get raw temperature
            v0 + c6             ; subtract 3a (temp @ 0 øC)
                                ; by adding c6
            p - a stack         ; point to a stack
            v1 = 18             ; scaling factor is 18h/53h
            v0 * v1 to mp       ; multiply by 18...
            v1 = 53
            v0 - mp/v1          ; and divide by 53
            v0 to 3dec mp       ; convert to decimals
            p + 1               ; point to tens
            ld b,c              ; load display
            ret
tempin1     asciz "Temp in  =    oC"
; note: change 6F (o) in hexfile into LCD degree sign
```

Listing 8-1: *De listing van Tempin.asm.*

De hardware van Chip Homesystem

In figuur 8-5 is de "hardware" van Chip Homesystem te zien. Op **output 0** is via een schakeltransistor een relais aangesloten voor sturing van een schemerlamp. Op **output 1** een identieke schakeling voor sturing van een alarmlamp en een luide buzzer.

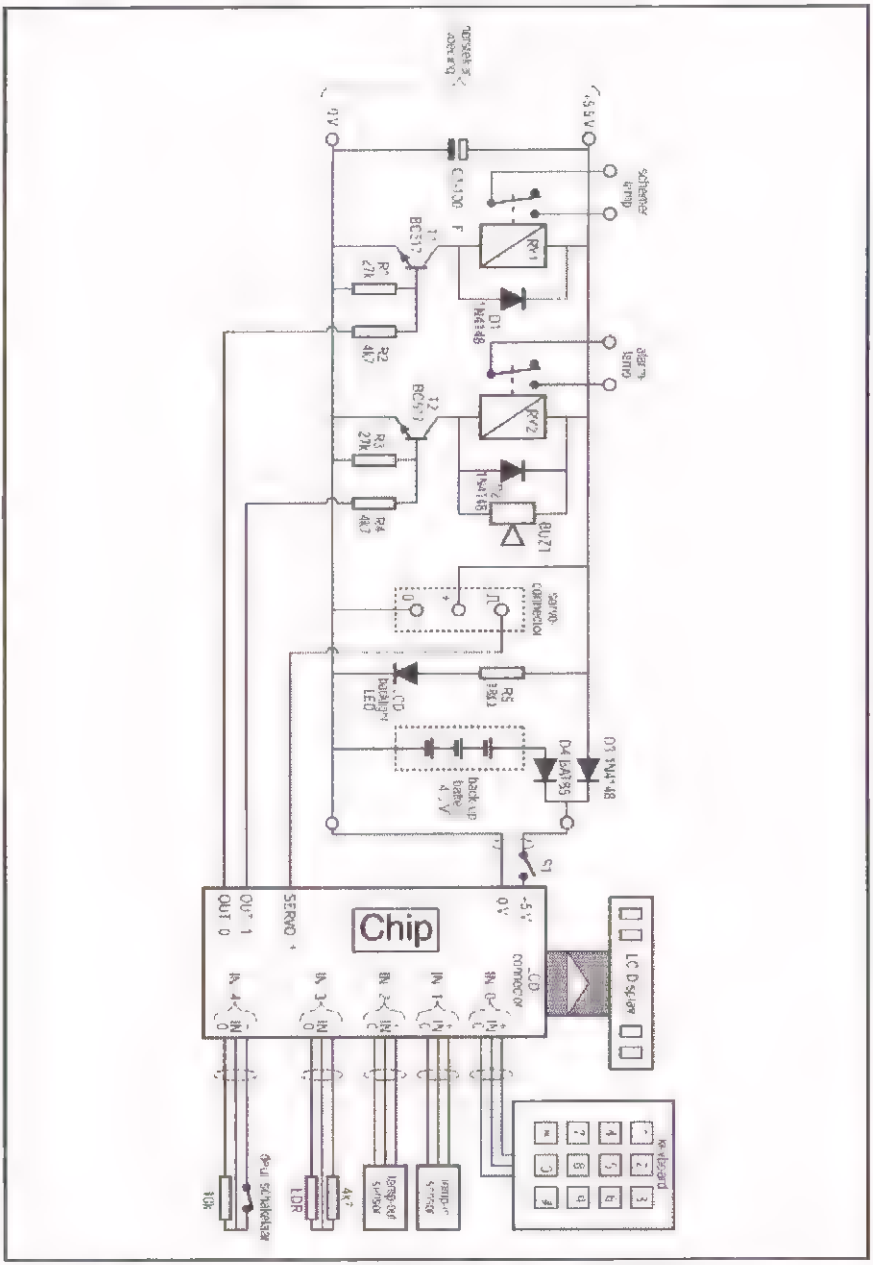
Tenslotte is **servo 1** beschikbaar om de kachel uit of aan te zetten. Het is niet mogelijk om voor de kachelbediening een standaard recept te geven omdat dat per geval kan verschillen. Bij ons bleek de draaiknop van de radiatorkraan een pal in te drukken of los te laten. Op de kraan hebben we een hefboomconstructie gemonteerd waarbij de servo via de hefboom de pal indrukt of loslaat. Belangrijk is wel, dat de pal door de servo helemaal wordt ingedrukt of losgelaten, maar dat de servo daarbij niet mechanisch tegen zijn eindblokkeringen oploopt. Dan zou de servomotor stroom blijven trekken en niet lang meegaan en ook kan de voeding worden overbelast.

```
; Listing 8.2: Tempout.asm
;
main        skip out f = 0
            call tempout
            jp main

; tempout, subroutine to measure exterior temperature with \
; kty 10 6
; chip temp-out sensor connected to input 2
;
tempout     res out f           ; reset seconds flag
            p = tempou2         ; point to text
            ld 0,f              ; load display
            v0 = ana 2          ; get raw temperature
            v0 + 98             ; subtract 68h (temp 0 øC) by
                                ; adding 98h
            skip vf = 00        ; skip if below 0 øC
            jp tempou1
            v1 = 00             ; v0 = abs(v0)
            v1 - v0             ; note: 00h - e0h = 20h
            v1 to v0
            p = tempou3         ; point to minus sign and ...
            ld a,a              ; load display
tempou1     p = a-stack         ; point to a-stack
            v1 = 19             ; scaling factor is 19h/3fh
            v0 * v1 to mp       ; multiply by 19...
            v1 - 3f
            v0 = mp/v1          ; and divide by 3f
            v0 to 3dec mp       ; convert to decimals
            p + 1               ; point to tens
            ld b,c              ; load display
            ret
tempou2     asciz "Temp out=     oC"
tempou3     asciz "-"
; note: change 6F (o) in hexfile into DF (LCD degree sign)
```

***Listing 8-2:** De listing van Tempout.asm.*

Alles, inclusief de backlight LED van het LCD, wordt gevoed door een gestabiliseerde netstekervoeding (Friwo EP2) die op 5,5 V is afgeregeld. De Chip homecomputer wordt normaal door de netstekervoeding gevoed via diode D3, maar bij uitval van de netspanning door de 4,5 V batterij via D4. Het backlight wordt via een serieweerstand van 10 Ω, 1 W direct door de netstekervoeding gevoed.
Om aardlussen te voorkomen zijn alleen de actieve pennen van out 0, out 1 en servo 1 aangesloten. De 0 V loopt via de voeding.

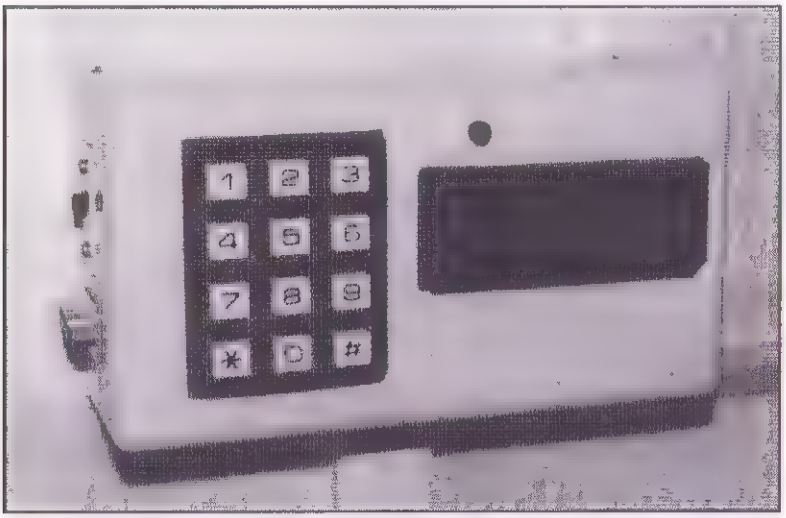


Figuur 8-5: *De volledige elektronica van het Chip Homesystem.*

Op **input 0** is het keyboard aangesloten, op **input 1** de sensor voor de binnentemperatuur, op **input 2** die voor de buitentemperatuur, op **input 3** een LDR die het buitenlicht meet en op **input 4** een buitendeur schakelaar, zo'n type met een reedswitch, dat door een magneet op de deur gesloten wordt gehouden en open gaat als de deur wordt geopend. In dat geval wordt input 4 naar laag getrokken door de 10 kΩ weerstand.

De behuizing van het systeem

Omdat wij ons Chip experimenteersysteem nodig hebben voor het realiseren van andere toepassingen, hebben we nog een Chip gebouwd en die samen met een LCD (met backlight LED) en een keyboardje netjes in een OKW standaardbox 1 met een lengte van 150 mm gemonteerd, zie figuur 8-6. De Chip print zit met twee zeskant afstandstukken op de bodem van het kastje vast, met tussen de bodem en de afstandstukken een aluminiumplaatje, dat door de metalen afstandstukken verbonden is met de massa van de Chip print. In de linkerzijkant van het kastje zijn een aan/uitschakelaar en een connector voor de seriële verbinding opgenomen. De homecomputer is op een goed bereikbare plaats op ooghoogte tegen de wand bevestigd. Recht eronder, iets boven de plint, is eenzelfde kastje voor de opname van aanvullende elektronica gemonteerd. Tussen de kastjes zit een kabelgoot en in de kastjes zijn uitsparingen gemaakt voor de bedrading via de kabelgoot.



Figuur 8-6: *De behuizing van het Chip Homesystem.*

Homesys software

Homesys.asm (listing 8-3) is in wezen een combinatie van listings die besproken zijn. Voor de aansturing van het display wordt een aparte teller gebruikt, die door **output f** eenmaal per seconde wordt verhoogd. Hierdoor is een "display verdeling" in de tijd verkregen. Eerst wordt gedurende drie seconden de actuele tijd getoond, dan gedurende een seconde elk, de binnentemperatuur, gevolgd door de buitentemperatuur en de gemeten waarde van het daglicht. Als de kachel aanstaat wordt de display cyclus besloten met **"Heater is active"**. Als basis voor het regelen van de kachel wordt een aparte variabele gebruikt **temp set**. De temperatuur die door de binnensensor is gemeten, wordt vergeleken met temp set. Als de temperatuur lager is wordt de kachel aangezet, is hij hoger, dan wordt hij uitgezet. Temp set wordt door drie tijdstrings geregeerd. 's Morgens vroeg wordt hij op 25 °C ingesteld, om 8 uur op 20 °C en 's avonds weer wat hoger.

Afhankelijk van de hoeveelheid buitenlicht én de tijd wordt de schemerlamp geregeld. 's Nachts is hij altijd uit, overdag wordt hij ingeschakeld als het buiten donker is. Dus als overdag zwarte wolken de lucht gaan bedekken, gaat de lamp aan.

Door middel van het keyboard kunnen instellingen worden "overruled" Door op * te drukken, komt men in eerste menukeuze **"Set temperature?"**. Door nogmaals op * te drukken verschijnt de volgende menukeuze **"Start egg-timer?"**, gevolgd door respectievelijk **"Set light on/of?"**, **"Set alarm on/of"** en **"Set date & time?"**. Bij deze laatste optie wordt direct geschreven naar registers in de microcontroller.

Een keuze wordt bevestigd door op # te drukken. Zolang deze ingedrukt blijft is de ingestelde waarde te zien. Na loslaten kan, waar toepasselijk, een waarde worden ingevoerd. De eierwekker kan worden ingesteld op maximaal 99 minuten en 99 seconden, zolang hij loopt verschijnt de resterende tijd op het display. Met menukeuze **"Set alarm on/of?"** kan het buitendeuralarm op scherp worden gezet. Na aanzetten heeft men vijf minuten de tijd om het huis te verlaten. Daarna zal het openen van de buitendeur het alarm doen afgaan.

"Set temperature" en **"Set light on/off"** zijn tijdelijke instellingen, namelijk tot de volgende van toepassing zijnde tijdstring geldig wordt.

Opmerking

Een belangrijk aspect bij de verwerking van de toetsinvoer is dat de hoofdlus van het programma altijd blijft doorlopen. Dit deel van het programma

is wat ondoorzichtig om direct te begrijpen, het opstellen van een stroomdiagram maakt alles veel duidelijker.
De ingestelde waarden en tijden zullen voor sommige lezers niet direct van toepassing zijn, maar dat is nu juist het aardige van dit systeem. Men kan alles aanpassen en het programma kan gemakkelijk worden uitgebreid. Op de buitenwacht maakt Chip als Homesystem in elk geval veel indruk.

```
; Listing 8.3: Homesys.asm
; program to automate the house-heating, house lighting,
; intruder alarm
; includes running clock with weeks, days, hours, minutes
; and seconds
; also an external temperature sensor and an eggtimer.
; new functions can easily be added.
;
; use of inputs, outputs and variables
;
; in 0     - keyboard
; in 1     = temp in sensor
; in 2     = temp out sensor
; in 3     = LDR sensor
; in 4     = door switch (alarm)
; out 0    - relay 1 = light
; out 1    - relay 2 = alarm
; out 4    = continuity check
; out 6    = rotate text is active
; out 7      waiting time flag before setting alarm
; out 8    - alarm on/off
; out 9      light automatic on/off
; out a    = heater on flag
; out b      eggtimer on flag
; out c    = wait-on flag
; out d    - key down flag
; out e    - select flag
; out f is set every second by the operating system
; v0-v4    - general purpose variables
; v7         light value low
; v8       - light value high
; v9         temp low
; va       = temp medium
; vb         temp high
; vc       - temp set
; vd         time distributor
; ve       = selector
; vf is carry/borrow/overflow flag
;
weekvar   equ 8aa          ; internal weeks register
;
```

```
start     set out 9          ; set light automatic on
          p = initvar
          v7,ve - mp         ; initialise v7 - ve
          son                ; start servo drive
main_0    set out 4          ; set out 4 for checking with scoop
          res out 4          ; reset out 4 for checking with scoop
          skip out 8 = 1     ; skip if alarm flag is on
          jp main_1
          skip in 4 = 0      ; skip if house entry door switch
                             ; activated
          jp main_1
          set out 1          ; activate alarm
          v0 = 02
          v0 to mintimer     ; load minutes timer
main 1    skip out d = 0     ; skip if .not. key down
          jp keydown         ; key beeing pressed, jump to keydown
          skip out e   0     ; skip if .not. select active
          jp select1         ; select active, jump to select
          skip ve <> ff      ; skip if ve .not. active (= ff)
          jp main_3          ; select .not. active, jump to
                             ; continue main loop
          jp getkeyn         ; select is active, try to get a
                             ; key value
main_2    skip ve <> 00      ; jump according to selector value ve
          jp settem1
          skip ve <> 10
          jp setegg1
          skip ve <> 40
          jp settim1
main 3    v0 = key 0         ; get a key value
          skip v0 <> 3a      ; skip if .not. 3a (= *)
          jp select0         ; got a key value 3a, jump to select0
          skip out f - 1     ; skip if on the seconds flag
          jp main_0          ; no seconds flag, loop to main_0
;
; distribution in time (every second)
;
          res out f          ; reset seconds flag
          call checkDT       ; check date-time strings and if
                             ; necessary take action
          skip out 6 = 0     ; skip if rotating text is off
          jp rotatin
secs_0    skip out b = 1     ; skip if eggtimer is active
          jp secs 1          ; jump to continue distribution
                             ; in time
          call eggshow       ; show eggtimer value
          jp main_0          ; and loop to main
secs_1    vd + 01            ; increment time distributor vd
          skip vd = 01       ; perform the task as set by vd
          jp secs_2
          p = clock2         ; point to the initial clock text
          ld 0,f             ; and load the initial clock display
```

```
            call clock          ; call the clock sub
            jp main 0
secs_2      skip vd = 02
            jp secs 3
            call clockSM        ; refresh clock seconds and minutes
            skip out 1 = 1      ; skip if alarm has been activated
            jp main_0
            v0 = mintimer       ; get minutes timer
            skip v0 <> 00       ; skip if not yet zero
            res out 1           ; zero, set alarm off
            jp main_0
secs 3      skip vd = 03
            jp secs_4
            call clockSM        ; refresh clock seconds and minutes
            skip out 7 = 1      ; skip if alarm activate waiting
                                ; time flag
            jp main 0
            v0 = mintimer       ; get minutes timer
            skip v0 = 00        ; skip if zero
            jp main_0           ; not yet zero
            res out 7           ; reset alarm activate waiting
                                ; time flag ...
            set out 8           ; ... and set alarm to sharp
            jp main_0
secs 4      skip vd = 04
            jp secs 5
            call clockSM        ; refresh clock seconds and minutes
            call birtday        ; check for birthday, if so,
                                ; congratulate
            jp main_0
secs_5      skip vd = 05
            jp secs_6
            call tempin         ; call show interior temperature
            jp main 0
secs 6      skip vd = 06
            jp secs_7
            call tempout        ; call show exterior temperature
            jp main 0
secs_7      skip vd = 07
            jp secs_8
            call daylite        ; call show indicative external
                                ; light value
            skip out a = 1      ; skip if heater on flag
            vd = 00             ; show clock again if heater is off
            jp main_0
secs 8      p = heatonT         ; else show heaton text
            ld 0,f
            vd = 00
            jp main_0
heatonT     asciz "Heater is active"
;
rotatin     v0 = mintimer       ; get minutes timer
```

```
          skip v0 = 00      ; skip if run out
          jp main_0         ; still running, jump to main
          res out 6         ; reset rotate active flag
          stop rotate       ; disable rotate and reset display
          vd = 00           ; set seconds sequencer to phase 0
          jp main_0         ; jump to normal sequence display
;
; keydown, jump routine to wait for key release or time out
;
keydown   v0 = key 0        ; v0 is key value
          skip v0 = 3f      ; skip if no key pressed
          jp keydwn1        ; jump, key still beeing pressed
          res out d         ; key has been released,
                            ; reset key down
          jp main_0         ; and jump to main_0
keydwn1   v0 = timer        ; get time out value
          skip v0 - 00
          jp main_0         ; time still running, jump to main_0
          res out c         ; reset wait on flag
          res out e         ; reset select flag
          vd   00           ; reset time distributor
          ve - ff           ; set selector ve to off
          v0   02           ; give a sound signal
          v0 to tone        ; while key beeing pressed
          jp main 0         ; and loop to main_0
;
; getkeyn, jump routine to get a numerical key value for passing
; to selected
; jump routine main_2 according to selector ve
;
getkeyn v0 - key 0          ; get a key value into v0
          v1 - 39
          skip v0 > v1      ; skip if .not. numerical (0-9)
          jp getkey1        ; jump, got numerical value
          v0 = timer        ; get time out value
          skip v0 - 00      ; skip on time out
          jp main_0         ; not yet time out, jump to try
                            ; to get good value
          ve = ff           ; set selector ve to off
          vd = 00           ; set time distributor to show
                            ; clock (load mask)
          jp main_0         ; and continue main task
getkey1 set out d           ; got key value, set key down
          v1 = 03
          v1 to tone        ; give key beep
          v1 - 8c           ; circa 5 seconds
          v1 to timer       ; set time out value
          jp main_2         ; jump to selected set routine
                            ; in main_2
;
; select, jump routine to make a choice out of rotating menus
;
```

```
select0   v0 = 02
          v0 to tone          ; give a key beep
          vd = 00             ; reset time distributor
;         vd to mintimer      ; set minutes timer to zero and ...
          res out 6           ; reset rotate text flag in case ...
          stop rotate         ; text was rotating
          res out b           ; disable eggtimer
          p = seltxt?         ; display the ?
          ld f,f              ; at position f
          p = seltxt0         ; set p to the first of the
                              ; selections
          ld 0,e              ; load chars 0 e
          ve = 00             ; preset the selector to 00
          set out e           ; set selection to active
          set out d           ; set key down to active
          v0 = 3c
          v0 to timer         ; load time out value
          jp main_0           ; jump back to main_0
select1   skip out c = 0      ; entry point for rotate selections
          jp select2          ; jump if time out value already
                              ; loaded
          v0 = 3c             ; c = 0, load time out value
          v0 to timer
          set out c           ; and set c
select2   v0 = key 0          ; get a key value
          skip v0 <> 3a
          jp select3          ; jump to select3 if key
                              ; value = 3a (*)
          skip v0 <> 3b
          jp select4          ; jump to select4 if key
                              ; value = 3b (#)
          v0 = timer          ; get time out value
          skip v0 = 00        ; skip if time is out
          jp main_0           ; not yet, keep trying to get
                              ; 3a or 3b
          res out e           ; time is out, reset selector
                              ; (no more active)
          res out c           ; reset wait on
          jp main_0           ; and loop to perform main tasks
select3   v0 = 02             ; got key value 3a (*)
          v0 to tone          ; give key beep
          skip ve <> 40       ; skip if .not. last selector value
          ve = f0             ; yes, last, preset to f0
          ve + 10             ; add 10 to selector value
                              ; (last will become 00)
          p = seltxt0         ; set pointer to first slection text
          p + ve              ; add selector offset to p
          ld 0,e              ; and put text on display
          res out c           ; reset wait on flag
          set out d           ; set keydown flag
          v0 = 3c
          v0 to timer         ; load time out value
```

```
          jp main_0          ; loop to cycle through selection
                             ; while .not. time out
select4   v0 = 02            ; got a key value 3b (#)
          v0 to tone         ; give key beep
          res out c          ; reset wait on
          res out e          ; reset selector flag (disable
                             ; jump to select)
          set out d          ; set key down flag
          v0 = 6c
          v0 to timer        ; set time out value
          skip ve <> 00      ; and jump to entry point of set
                             ; routines
          jp settemp         ; according to selector value ve
          skip ve <> 10
          jp seteggt
          skip ve <> 20
          jp setlite
          skip ve <> 30
          jp setalar
          skip ve <> 40
          jp settime
          break              ; never reach this point
seltxt0   asciz "Set temperature"  ; selector ve = 00
          asciz "Start egg timer"  ; selector ve = 10
          asciz "Set light on/of"  ; selector ve = 20
          asciz "Set alarm on/of"  ; selector ve = 30
          asciz "Set date & time"  ; selector ve = 40
seltxt?   asciz "?"
;
; setalar, jump routine to set alarm on or off
;
setalar   skip out 8 = 1     ; skip if alarm on
          skip out 7 = 0     ; skip if waiting time flag
          jp setala1         ; jump to set alarm off
          set out 7          ; set alarm waiting time flag on
          v0 = 05
          v0 to mintimer     ; set waiting time before setting
                             ; alarm to 5 minutes
          p = setala4        ; point to text on
          jp setala2
setala1   res out 7          ; set waiting time flag off
          res out 8          ; set alarm flag to off
          res out 1          ; set alarm off
          v0 = 00
          v0 to mintimer     ; clear minutes timer
          v0 to sectimer
          p = setala3        ; point to text off
setala2   ld 0,f             ; put on display
          ve = ff            ; set selector ve to off
          vd = 00            ; reset time distributor (to load
                             ; clock mask and clock)
          jp main_0          ; jump to main task
```

```
setala3   asciz "alarm = set off!"
setala4   asciz "alarm = set on! "
;
; setlite, jump routine to set light on or off
;
setlite   skip out 0 = 0   ; skip if light off
          jp setlit1       ; jump to set light off
          res out 9        ; set light automatic to off
          set out 0        ; set light on
          p = setlit4      ; point to text on
          jp setlit2
setlit1   res out 9        ; set light automatic to off
          res out 0        ; set light off
          p = setlit3      ; point to text off
setlit2   ld 0,f           ; put on display
          ve = ff          ; set selector ve to off
          vd = 00          ; reset time distributor (to
                           ; load clock mask)
          jp main 0        ; jump to main task
setlit3   asciz "light = set off!"
setlit4   asciz "light = set on! "
;
; seteggt, jump routine to load the eggtimer
;
seteggt   p = setegg4      ; entry point from select, point
                           ; to mask
          ld 0,f           ; display mask
          p = a stack      ; point p to a-stack
          v0 = 30          ; and load hundreds with 30 - ascii 0
          v0,v0 to mp
          p + 1            ; set p to tens
          v2 = 09          ; set v2 as index to tens of minutes
          jp main_0        ; jump to main to get a key value
                           ; (or time out)
setegg1   v0,v0 to mp      ; load key value into mp + 1 (tens)
          ld v2,v2         ; put tens on display according
                           ; index v2
          p + 1            ; increment pointer
          v2 + 01          ; increment index v2
          skip v2 <> 0b
          jp setegg2       ; jump if index v2 = 0b
          skip v2 <> 0f
          jp setegg3       ; jump if index v2 = 0f
          jp main_0        ; not 0b, not 0f, jump to main to
                           ; get next key value
setegg2   p = a-stack      ; index = 0b,
          v3 = 3dec mp     ; save minutes in v3
          p + 1            ; point p to tens
          v2 = 0d          ; adjust index v2 to tens of
                           ; seconds on display
          jp main_0        ; jump to main to get next key values
setegg3   p = a-stack      ; index = 0f
```

```
          v0 = 3dec mp      ; convert key values to byte into v0
          v0 to sectimer    ; and load into seconds timer
          v3 to mintimer    ; load minutes timer
          ve = ff           ; set selector ve to off
          set out b         ; set out b as flag for eggtimer
                            ; loaded
          jp main_0         ; resume main task
setegg4   asciz "Eggtime: ??m ??s"
;
; settemp, jump routine to display/set the 'set' temperature
;
settemp   p = settem2       ; entry point from select, point
                            ; to mask
          ld 0,f            ; display mask
          p = a-stack       ; convert vc (= set temperature)
          vc to 3 dec mp    ; to decimal on the a stack
          p + 1             ; point to tens (hundreds will be
                            ; 30h = ascii 0)
          ld b,c            ; display tens and units
          v2 = 0b           ; set v2 as index to display 0b
                            ; (tens of minutes)
          jp main_0         ; jump to main to get key value
                            ; (or time out)
settem1   v0,v0 to mp       ; load key value into mp + 1 (tens)
          ld v2,v2          ; load on display via index v2
          p + 1             ; point to units
          v2 + 01           ; point index v2 to units
          skip v2 = 0d      ; skip if we got tens and units
          jp main_0         ; not yet ready, jump back to main
          p = a-stack       ; got two values
          vc = 3dec mp      ; convert to byte and load into
                            ; vc (new 'set' temp)
          ve = ff           ; set selector ve to off
          vd = 00           ; reset time distributor (to load
                            ; clock mask)
          jp main 0         ; jump to main task
settem2   asciz "Temp set =    °C"
; note: change 6F (o) in hexfile into DF (LCD degree sign)
;
; settime, jump routine to display/set the date & time
;
settime   p = settimT       ; point to date/time mask
          ld 0,f            ; put on display
          p = a-stack       ; point p to a-stack
          v0 = 30           ; and load hundreds with 30 = ascii 0
          v0,v0 to mp
          p + 1             ; set p to tens
          v2   00           ; nibble counter = 00
          v3 = 00           ; display index counter = 00
          v4   00           ; byte index counter = 00
          jp main 0
settim1   v0,v0 to mp       ; load key value into mp + 1 (tens)
```

```
          v2 + 01          ; increment nibble counter
          v3 + 01          ; increment display index counter
          ld v3,v3         ; load on display via index v2
          p + 1            ; point to units
          v0 = 01          ; is the nibble counter odd
          v0 and v2        ; get only bit 0
          skip v0 = 00     ; skip if even
          jp main_0        ; yes, odd get low nibble
          p = a stack
          v0 = 3dec mp     ; convert to byte
          p = settimS      ; point to start of temporarily
                           ; storage place
          p + v4           ; point to storage position
          v0,v0 to mp      ; store byte
          v4 + 01          ; increment byte index counter
          v3 + 01          ; increment display index counter
          p = a stack
          p + 1            ; point to tens on a-stack
          skip v4 = 05     ; skip if we got 5 bytes
          jp main_0
          p   settimS      ; point to start of storage
          v0,v4 = mp       ; copy bytes into v0-v4
          p = weekvar      ; point to clock weeks internal
                           ; register
          v0,v4 to mp      ; copy variables into internal regs
          ve = ff          ; set selector ve to off
          vd = 00          ; reset time distributor (to load
                           ; clock mask)
          jp main_0        ; jump to main task
settimT   asciz " ??:??:??:??:?? "    ; date & time mask
settimS   bytes 0011223344 ; temporarily storage place for bytes
;
; subroutine checkDT, compares date-time strings against the
; running clock
; and takes the necessary action when equal.
;
checkDT   p = DTliteS      ; point to set light automatic on
                           ; DT-string
          call dattime     ; compare
          skip vf = 00     ; skip not equal
          set out 9        ; set light automatic flag on
          p = DTliteR      ; point to set light automatic off
                           ; DT-string
          call dattime     ; compare
          skip vf <> 00    ; skip if equal
          jp checkD1       ; jump next
          res out 9        ; set light automatic flag off
          res out 0        ; set light off
checkD1   p = DTtempH      ; point to set temperature high
                           ; DT string
          call dattime     ; compare
          skip vf = 00     ; skip not equal
```

```
            vb to vc            ; copy high temperature into temp set
            p = DTtempM         ; point to set temperature medium
                                ; DT-string
            call dattime        ; compare
            skip vf = 00        ; skip not equal
            va to vc            ; copy medium temperature into
                                ; temp set
            p = DTtempL         ; point to set temperature low
                                ; DT-string
            call dattime        ; compare
            skip vf = 00        ; skip not equal
            v9 to vc            ; copy low temperature into temp set
            ret
;
; Date Time string to set light automatic on, every day at 06.00

;
DTliteS     bytes 7f7f          ; weeks, days; note: 7f = does't care
            bytes 0600          ; hours, minutes
            bytes 7f            ; seconds
;
; Date Time string to set light automatic off, every day
; at 19.00
;
DTliteR     bytes 7f7f          ; weeks, days
            bytes 1300          ; hours, minutes
            bytes 7f            ; seconds
;
; Date Time string to set temperature to high, every day
; at 05.30
;
DTtempH     bytes 7f7f          ; weeks, days
            bytes 051e          ; hours, minutes
            bytes 7f            ; seconds
;
; Date Time string to set temperature to medium, every day
; at 08.00
;
DTtempM     bytes 7f7f          ; weeks, days
            bytes 0800          ; hours, minutes
            bytes 7f            ; seconds
;
; Date Time string to set temperature to low, every day at 18.30

;
DTtempL     bytes 7f7f          ; weeks, days
            bytes 121e          ; hours, minutes
            bytes 7f            ; seconds
;
; birtday, subroutine to show birthday rotating text on display
; because weeks and days are used these must be adjusted
; every year
```

```
birtday   p = DTbirt1      ; point to birthday date-time 1
          call dattime     ; compare
          skip vf <> 00    ; skip if equal
          ret              ; else return
          set out 6        ; set rotating flag
          p = birttx1      ; point to birthday text
          rotate           ; start rotating
          v0 - 05
          v0 to mintimer   ; for 5 minutes
          ret
;
birttx1   asciz "Congratulations Bob!"
;
DTbirt1   bytes 2b02       ; weeks, days; Week 43, day 2,
                           ; 08:10:00
          bytes 080a       ; hours, minutes
          bytes 7f         ; seconds
;
; eggshow, subroutine to show minutes and seconds timer
; the eggtimer mask has already been displayed by seteggt
;
eggshow   v0 - mintimer    ; get minutes into v0
          p - a stack
          v0 to 3dec mp    ; convert to decimal on the a-stack
          p + 1            ; point to tens
          ld 9,a           ; display tens and units
          v1 - sectimer    ; get seconds into v1
          p = a-stack
          v1 to 3dec mp    ; convert to decimal on the a-stack
          p + 1            ; point to tens
          ld d,e           ; display tens and units
          v0 or v1         ; v0 - v0 .or. v1
          skip v0 = 00     ; skip if both zero
          ret              ; not yet zero, return
          res out b        ; reset eggtimer active flag
          vd - 00          ; set time distributor to 00
                           ; (load clock mask)
          v0 = ff          ; alert user that
          v0 to tone       ; time has run out
          v0 = 03          ; because the beeper is not always
                           ; heard
          v1 = 01          ; we use the alarm to notify the user
          v0 to sectimer
          v1 to mintimer
          set out 1
          ret
;
; clock, subroutine to show the week, day of week and time
; on the LCD.
; Note that we use variables to load the display,
; so that we can use a loop.
;
```

```
clock     v0   01          ; initialise v0 and v1
          v1 - 02
clock1    skip v0 <> 01     ; get the corresponding byte
          v3 = weeks        ; and load into v3
          skip v0 <> 04
          v3 = days
          skip v0 <> 07
          v3 - hours
          skip v0 <> 0a
          v3 = minutes
          skip v0 <> 0d
          v3 - seconds
          p - a stack       ; we let p point to a-stack
          v3 to 3dec        ; and convert v3 to 3 decimals
          p + 1             ; we don't need the hundreds
          ld v0,v1          ; show tens and units on display
          v0 + 03           ; let v0 and v1 point
          v1 + 03           ; to next display position
          skip v0 = 10      ; if v0 = 10 we are ready
          jp clock1         ; else we loop
          ret
clock2    asciz "   :  :  :  :   "   ; this is the inititial
                                      ; display
;
clockSM   v0 = seconds      ; this short routine refreshes only
          p - a stack       ; the seconds and minutes, in order
          v0 to 3dec mp     ; to save time
          p + 1             ; probably nobody will notice
                            ; the hours...
          ld d,e
          v0 - minutes
          p = a-stack
          v0 to 3dec mp
          p + 1
          ld a,b
          ret
;
; tempin, subroutine to measure interior temperature with
; kty 10 6.
; chip sensor connected to input 1.
; Midpoint is 20 øC, so the range is 0-40 øC. At 20 øC Rkty
; 10-6 = 1926 ê
; The formulae for the temperature in øC: (raw - 3dh) * 18h/53h
; (see curve).
;
tempin    p = temitxt       ; set pointer to temperature in mask
          ld 0,f            ; display mask
          v0 = ana 1        ; get raw temperature
          v0 + c3           ; subtract 3dh (temp @ 0 øC) by
                            ; adding c3h
          p   a stack       ; set p to a-stack
          v1 - 18           ; and
```

```
          v0 * v1 to mp     ; multiply raw by 18h
          v1 = 53           ; then
          v0 = mp/v1        ; divide by 53h
          v0 to 3dec mp     ; convert to decimal
          p = 1             ; set p to tens
          ld b,c            ; display temp in øC
;
; check measured temperature (ist value) against set temperature
; (sol_value),
; when ist_val < sol_val then set heater on,
; when ist val > sol_val then set heater off
; when ist_val = sol val then do nothing (hysteresis)
;
          skip v0 <> vc     ; skip ist_val <> sol_val
          ret               ; equal, return nothing to do
          v0 - vc           ; ist_val = ist_val - sol_val
          skip vf = 00      ; skip if ist val > sol val
                            ; (vf = borrow)
          jp tempin1        ; ist_val < sol_val, jump to set
                            ; heater on
          call resheat      ; call sub to set heater off
          ret
tempin1   call setheat      ; call sub to set heater on
          ret
temitxt   asciz "Temp in =     °C"
; note: change 6F (o) in hexfile into DF (LCD degree sign)
;
; setheat and resheat, subroutines to set heater on if not
; already on,
; or off if not already off, the heater motor is servo 1
; out a   = heater on flag
;
setheat   skip out a = 0    ; skip if heater is off
          ret               ; already on, return
          set out a         ; set heater on flag
          v0 = ff
          v0 to s1          ; and rotate heater servo
          ret
;
resheat   skip out a = 1    ; skip if heater is on
          ret               ; already off, return
          res out a         ; reset heater on flag
          v0 = 0a
          v0 to s1          ; and rotate heater servo
          ret
;
; tempout, subroutine to measure exteriour temperature with
; kty 10-6
; chip sensor connected to input 2
; Midpoint is 10 øC, so the range is -20 - +40 øC. At 10 øC
; Rkty 10-6 = 1783 ê
; The formulae for the temperature in øC: (raw - 68h) * 19h/3fh
```

```
; (see curve).
;
tempout   p = temotxt      ; set pointer to temperature out mask
          ld 0,f           ; display mask
          v0 - ana 2       ; get raw temperature
          v0 + 98          ; subtract 68h (temp @ 0 øC) by
                           ; adding 98h
          skip vf = 00     ; skip on no carry
          jp tempou1       ; there was a carry, so jump
          v1 = 00          ; the temperature is below zero øC
          v1 - v0          ; get the negative temperature by
                           ; subtracting from 00
          v1 to v0         ; save into v0
          p = tempmin      ; set pointer to minus sign
          ld a,a           ; put on display
tempou1   p = a-stack      ; set pointer to a-stack
          v1 = 19          ; and
          v0 * v1 to mp    ; multiply raw by 19h
          v1 = 3f          ; then
          v0 = mp/v1       ; divide by 3fh
          v0 to 3dec mp    ; convert to decimal
          p + 1            ; point to tens
          ld b,c           ; and display temperature in øC
          ret
temotxt   asciz "Temp out=  ?? ²C"
tempmin   asciz "-"
; note: change 6F (o) in hexfile into DF (LCD degree sign)
;
; daylite, subroutine to measure and display daylight value \
; from LDR
; LDR connected to input 3
; the value is only indicative of the light
;
daylite p = daylit2        ; point to text
          ld 0,f           ; display text
          v0 = ff          ; ff = xor mask
          v1 = ana 3       ; v1 = light value
          v0 xor v1        ; take complement of light value
          p - a stack
          v0 to 3dec mp    ; convert to decimal
          ld d,f           ; put on display
          skip out 9 = 1   ; skip if light automatic is on
          ret              ; not on, return
;
; check measured light value against light value low (v7)
; and high (v8)
; when measured < low then set light on,
; when measured > high then set light off
;
          v0 to v1         ; copy light value into v1
          v0 - v7          ; v0 = measured   low
          skip vf = 00     ; skip if v0 > v7
```

```
          jp daylit1        ; jump to set light on
          v1 - v8           ; v1 = measured - high
          skip vf = 00      ; skip if v1 > v8
          ret               ; do nothing: low > measured >= high
          res out 0         ; set light off
          ret
daylit1   set out 0         ; set light on
          ret
daylit2   asciz "Lightvalue =     "
;
; dattime, subroutine to compare the running clock against a
; date-time string
; pointed to by the pointer:
; p -> weeks, days, hours, minutes, seconds (all values hex!)
; when a dattim string byte is 7f, it will not be tested
; when the condition is met for the first time, vf will be
; set as flag (<> 00)
; and bit 7 of weeks, date time string will be set. As soon
; as the condition
; is no more valid this bit wil be reset.
;
dattime   v0,v4 = mp        ; move string date and time into
                            ; v0...v4
          vf = 7f           ; strip bit 7 from v0
          v0 and vf
          skip v0 <> 7f     ; skip if weeks <> 7f
          jp dattim1        ; weeks = 7f, so do not test
          vf - weeks        ; get clock weeks
          skip v0 = vf      ; skip if equal
          jp dattim7        ; not equal, jump
dattim1   skip v1 <> 7f     ; skip if days <> 7f
          jp dattim2        ; days = 7f, so do not test
          vf = days         ; get clock days
          skip v1 = vf      ; skip if equel
          jp dattim7        ; not equal, jump
dattim2   skip v2 <> 7f     ; skip if hours <> 7f
          jp dattim3        ; hours = 7f, so do not test
          vf = hours        ; get clock hours
          skip v2 = vf      ; skip if equal
          jp dattim7        ; not equal, jump
dattim3   skip v3 <> 7f     ; skip if minutes <> 7f
          jp dattim4        ; minutes = 7f, so do not test
          vf = minutes      ; get clock minutes
          skip v3 = vf      ; skip if equal
          jp dattim7        ; not equal, jump
dattim4   skip v4 <> 7f     ; skip if seconds <> 7f
          jp dattim5        ; seconds = 7f, so do not test
          vf = seconds      ; get clock seconds
          skip v4 = vf      ; skip if equal
          jp dattim7        ; not equal, jump
;
; dattim string is equal to the clock
```

```
; when this happens for the first time, bit 7 of v0 (weeks)
; will be 0,
; it must be set to 1 and vf must be made <> 00
;
dattim5   v0, v0 = mp
          vf - 80          ; vf is bitmask
          vf and v0        ; strip d6...d0
          skip vf - 00     ; skip if d7 - 0
          jp dattim6       ; jump, this dattim skip has already
                           ; been taken
          vf = 80          ; vf is .or. bit 7
          v0 or vf         ; set 7 of v0
          v0, v0 to mp     ; write back to dattim string
          skip a           ; skip always, let vf remain 80
dattim6   vf = 00          ; entry point for skip already taken,
                           ; reset flag
          ret
dattim7   v0, v0 to mp     ; write back to dattim string
          vf = 00          ; reset flag vf
          ret
;
          org 7f0
; preset values are loaded at start-up
;         v7 = 64          ; light value low= 100d
;         v8 = 82          ; light value high= 130d
;         v9 = 10          ; temp low= 16d øC
;         va = 14          ; temp med= 20d øC
;         vb = 16          ; temp high= 22d øC
;         vc = 14          ; 'set' temperature= 20d øC
;         vd = 00          ; seconds sequencer= 00
;         ve = ff          ; selector= ff (idle)
;
;              v7v8v9vavbvcvdve
initvar   bytes 648210141614400ff
```

Listing 8-3: *Het complete Homesys.asm programma voor de automatisering van een huis.*

9 Chip als accutester

Inleiding

Om Chip als accutester te gebruiken zijn enkele aanvullende hardware componenten nodig. In de eerste plaats natuurlijk het keyboard voor de gegevensinvoer en verder een printje met de ontlader. Een heel aardig aspect is, dat gebruik kan worden gemaakt van hulpprogrammaatjes om de hardware te testen.
Deze NiCad-ontlader kan de capaciteit meten van zowel losse cellen als accupakketten met een open spanning tot 35 V. Meestal zullen dat oplaadbare cellen zijn, maar door de ruime instellingsmogelijkheden kunnen alle soorten cellen worden gemeten, ook niet-oplaadbare, mits de ontladen celspanning tussen 0,5 V en 2,5 V ligt. Tijdens de ontlading wordt iedere minuut de momentele spanning in het geheugen opgeslagen, zodat na afloop van het ontlaadproces de spanningscurve kan worden getekend.

Gegevens accutester

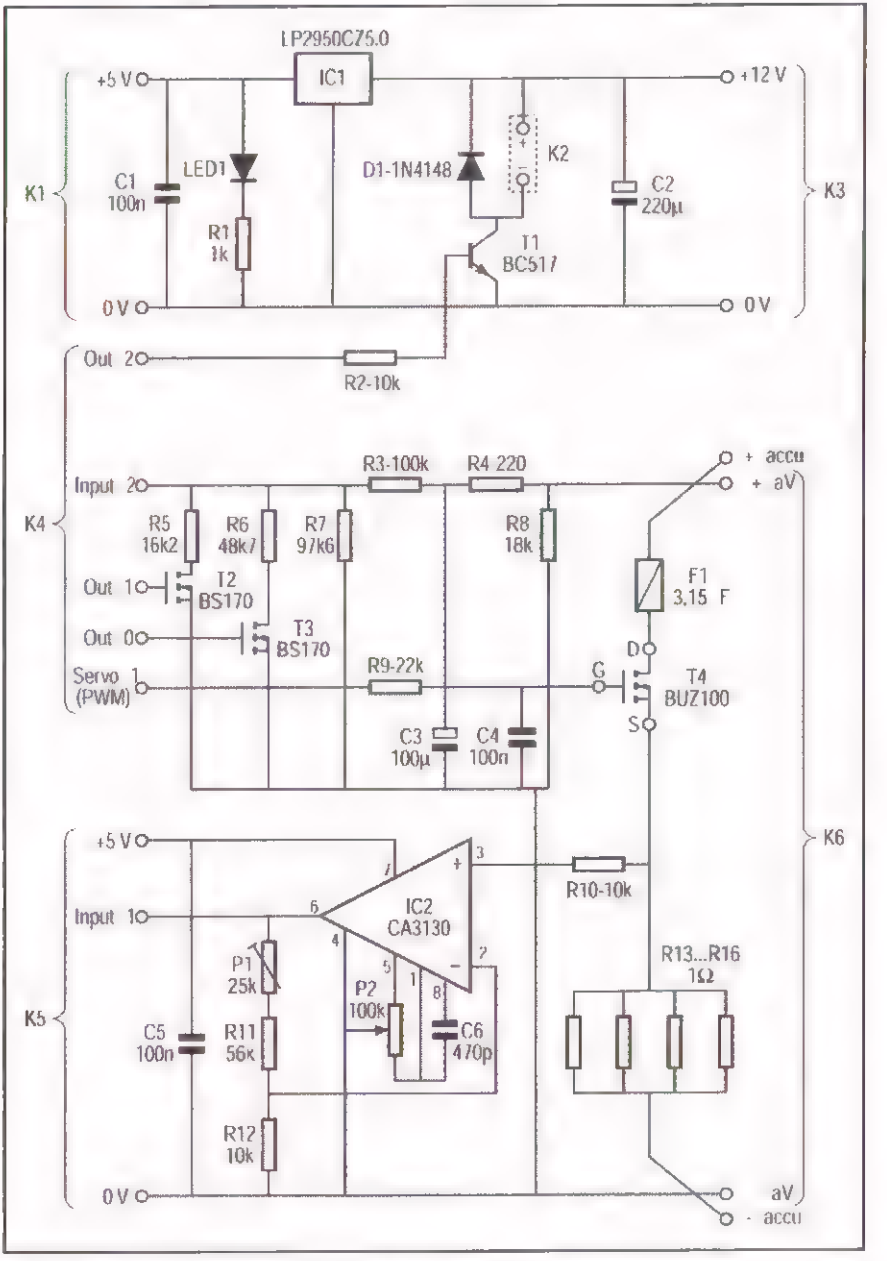
De gegevens van deze schakeling kunnen als volgt worden samengevat.

- ontlaadstroom:
  100-2500 mA per 10 mA
- ontlaadspanning:
  0,5-2,5 V/cel
- aantal cellen:
  1-20
- maximale ingangsspanning:
  35 V
- maximale dissipatie:
  60 W
- maximale aflezing:
  9.999 mAh
- opslag van instellingen en metingen:
  in EEPROM
- sample geheugen:
  508 bytes
- sample interval:
  1 minuut

- max. sampling tijd:
  8 uur 27 minuten
- max. elapsed time:
  24 u 59 m 59 s
- uitlezing:
  16 karakter LCD
- gegevensinvoer:
  12-toets keyboard
- communicatie met PC:
  RS232, 19.200,n,7,2

Het schema van de ontlader

De ontlader, waarvan het schema in figuur 9-1 is getekend, bestaat uit drie delen. Het bovenste deel dient voor de spanningsverzorging en het aan- of uitschakelen van de ventilator. Voor de voeding is een ongestabiliseerde stekkervoeding gebruikt met omschakelbare uitgangsspanning: 12-9-7,5-6-4,5-3 V. De stabilisator (IC1) is een low drop type, dat al bij 6 V ingangsspanning precies 5 V levert. Door de uitgangsspanning van de voeding om te schakelen (6-12 V) kan de snelheid van de ventilator worden geregeld. De netvoeding wordt aangesloten op K3, de ventilator op K2 en op K1 wordt de voedingsconnector van Chip gezet. De spanning van de accu wordt gemeten op connector K6 (+aV en -aV).
Via een snoertje wordt deze connector direct verbonden met de apparaatklemmen, zodat de spanningsval over de zekering en stroomvoerende bedrading het meetresultaat niet beïnvloedt. Het signaal op K6 wordt gefilterd door R4 en C3 en vervolgens door een instelbare verzwakker met drie bereiken gevoerd: 10,2 V, 20,4 V en 40,8 V. De eindwaarde van ieder bereik levert FFh (255d). De verzwakker wordt ingesteld met **Out0** respectievelijk **Out1**. Door de keuze van de verzwakkerwaarden wordt (overbodig) rekenwerk voorkomen.
De rest van de schakeling dient voor de instelling en handhaving van de ontlaadstroom. Op uitgang **Servo1** staat een PWM-signaal dat door R9 en C4 wordt gefilterd en aan de gate van de power-MOSFET (T4) wordt toegevoerd.
In de source-aansluiting zijn de stroomsensor weerstanden R13-R16 opgenomen. De spanning hierover wordt door IC2 versterkt en via **Input1** op K5 naar Chip gevoerd. Als versterker is een op-amp gebruikt waarvan de uitgangsspanningszwaai tot de voedingsspanningen reikt. Met P1 wordt de versterking ingesteld en met P2 de offsetcompensatie.



Figuur 9-1: *Het schema van de accu ontlader.*

ONDERDELENLIJST

WEERSTANDEN, ¼ W, 5 % (behalve anders aangegeven)

R1	1 kΩ
R2,R10,R12	10 kΩ
R3	100 kΩ 1 %
R4	220 Ω
R5	16,2 kΩ 1 %
R6	48,7 kΩ 1 %
R7	97 6 kΩ 1 %
R8	18 kΩ
R9	22 kΩ
R11	56 kΩ
R13-R16	1 Ω metaalfilm 1/2 W, (2,5 x 6,3 mm)

INSTELPOTENTIOMETERS

P1	25 kΩ Piher 6 mm staand
P2	100 kΩ Piher 6 mm staand

CONDENSATOREN

C1,C4,C5	100 nF 50 V multilayer, RM 2.54
C2	220 µF 16 V (8,5 x 13 mm, RM 2,5 of 3,5)
C3	100 µF 35 V (8,5 x 13 mm, RM 2,5 of 3,5)
C6	470 pF 63 V ceramisch, RM 2,54

HALFGELEIDERS

D1	1N4148
LED1	rode LED 3 mm, high int.
T1	BC517
T2,T3	BS170 of BSN10A
T4	BUZ100
IC1	LP2950CZ5.0 Conrad best. nr. 17-56-76
IC2	CA3130

DIVERSEN

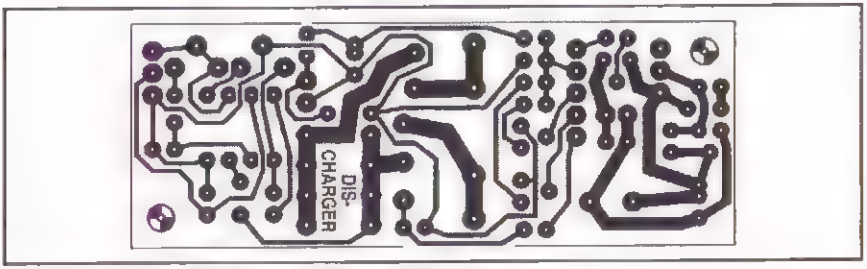
K1,K5	3-polige SIL printheader RM 2,54
K2,K6	2-polige printheader RM 2,54
K3	printkroonsteen steek RM 5
K4	5-polige SIL printheader RM 2,54
F1	zekering 3.15 A snel 20 x 5 mm
1	stel printzekeringhouders
1	IC-voet 8-pens
5	printpennen 1,3 mm + contra (insteekschoenen)

Voor de regeling van de PWM-werkslag wordt een variabele gebruikt, waaraan we kunnen zien of de stroom stabiel is. Het is namelijk mogelijk, dat de MOSFET niet in staat is om de vereiste stroom te leveren als de spanning lager is 1 V. De stroomvariabele "gaat dan door nul" en dit wordt dan gemeld.

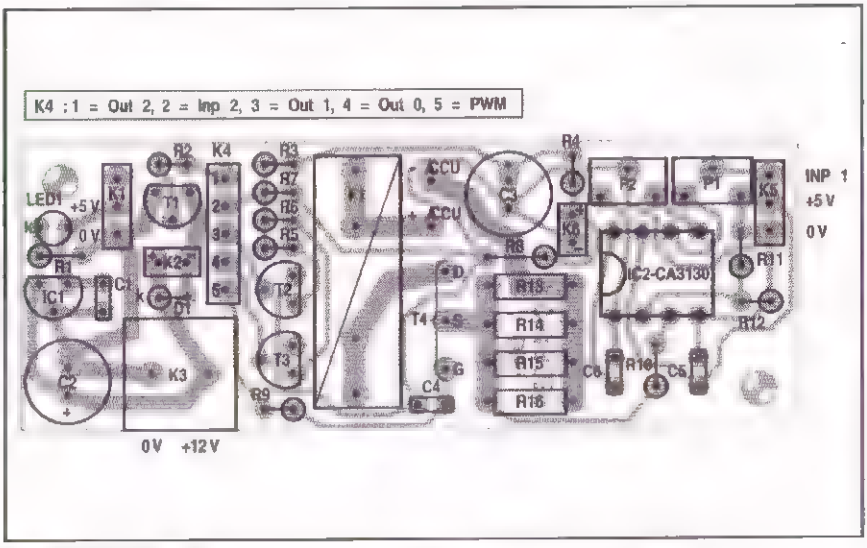
Tussen de drain en de source van de MOSFET bevindt zich een anti-parallel geschakelde bodydiode. Als een accu per abuis verkeerd om wordt aangesloten, brandt zekering F1 door en blijft de schade beperkt. Zaak is wel om een snelle zekering te gebruiken.

Bouw van de print

De print layout en onderdelenopstelling van de ontlader zijn in figuur 9-2 en 9-3 te zien. Ook nu is de print niet op ware grootte afgedrukt. Ga weer naar www.vego.nl/chip en open 09_02.tif in bijvoorbeeld PaintShop. Print het ontwerp af met als afmetingen 70 x 26 mm² op transparante folie.



Figuur 9-2: *De print voor de accutester.*

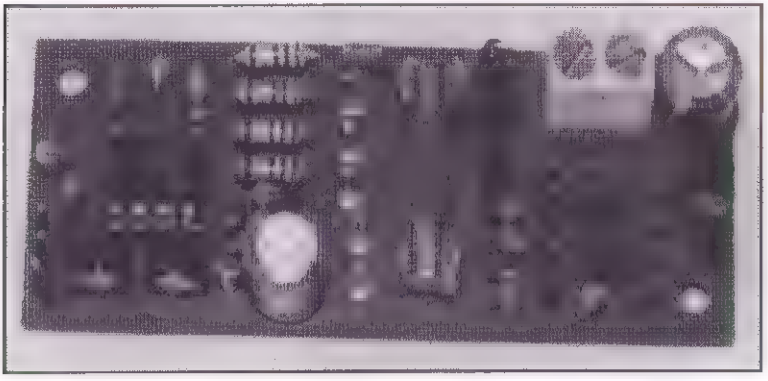


Figuur 9-3: *De componentenopstelling van de accutester.*

De bouw van het printje wijst zichzelf, voor alle connectors zijn male printheaders gebruikt, behalve voor K3 dat een printkroonsteen is en voor de stroomvoerende aansluitingen waarvoor printpennen van 1,3 mm zijn gebruikt. Voor opname van de zekering dienen twee metalen houdertjes. Voor IC2 is een voetje gebruikt. Weerstanden R13-R16 zijn 4 mm boven de print gemonteerd in verband met warmte-afvoer.
Op de printbanen +ACCU, -ACCU, D en S zijn soldeerrupsen gelegd om de weerstand te verminderen.

Het eindresultaat

In figuur 9-4 is het prototype van de print te zien.

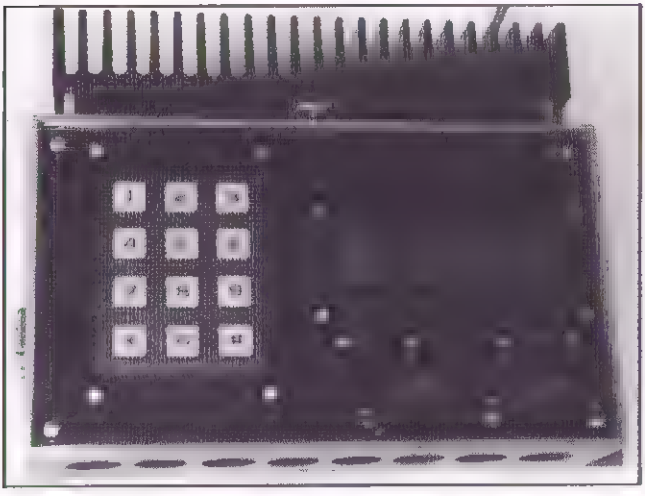


Figuur 9-4: *Het prototype van de print.*

Montage in een behuizing

De belangrijkste werkzaamheden voor de samenbouw betreffen de aanpassingen aan het Teko-kastje. De foto's in de figuren 9-5 en 9-6 geven een goed beeld van de opstellingen van de componenten in het kastje en de uitvoering van het frontplaatje. Voor een goed resultaat is passen, meten en secuur aftekenen van essentieel belang.
Zoals uit de foto blijkt is het uiterlijk van de NiCad-ontlader eenvoudig en overzichtelijk. Voor de invoer van de gegevens is een toetsenbordje aanwezig met 12 toetsen. Alle informatie verschijnt op het 16-karakter LCD. De accu wordt aangesloten op de twee instrumentklemmen, een zwarte en een rode. Verder zijn nog twee schuifschakelaars aanwezig, een voor aan/uit, op de ander komen we nog terug.
De ontlader kan prima als stand-alone apparaat werken, maar zoals zal blijken is gebruik van een PC in sommige gevallen onontbeerlijk. Daar-

voor is in de linker zijkant een negenpolige connector opgenomen, die kan worden verbonden met de seriële poort van de PC.
In de achterwand komt een grote opening voor de luchtdoorlaat van de ventilator en in de voorkant zitten acht gaten van 14 mm voor de inlaat van de koellucht. Wijs geworden door een mislukte poging met een metaalboor, hebben we dat gedaan met een langzaam draaiende houtboor, waarvan de snijkanten eerst zijn "ontscherpt" om te zorgen dat ze niet "happen".



Figuur 9-5: *De accutester ingebouwd in een mooie behuizing.*

ONDERDELENLIJST BEHUIZING

1	. . . . .	Teko lessenaar kastje type 362
1	. . . . .	koelelement Fischer SK-132/37,5/SA
1	. . . . .	ventilator 40 x 40 mm 12 Vdc
1	. . . . .	netadapter 300 mA 3-4,5-6-7,5-9-12 Vdc, ongestabiliseerd
1	. . . . .	apparaatklem rood bv. Hirschmann PKI-100
1	. . . . .	apparaatklem zwart idem
1	. . . . .	keyboard 12 toetsen met common-aansluiting en 12 individuele aansluitingen bv. Display best. nr. 03.52.1252 of Conrad best.nr. 19-55-61
1	. . . . .	miniatuur schuifschakelaar 2x om, soldeerogen, lengte 23 mm
1	. . . . .	strip female printheaders RM 2,54

Chip is op een aluminiumplaatje van 1,5 mm gemonteerd, met dezelfde afmetingen als de print, op afstandbussen van 5 mm. We hebben boutjes

met verzonken kop gebruikt zodat aan de onderzijde van het metalen plaatje geen uitsteeksels zijn.
Het maken van de opening voor de RS232-connector in de linker zijkant lijkt moeilijker dan het is. Van een boutje, dat in de schroefmoer van de RS232-connector past, is de kop afgezaagd en in een sneldraaiende boormachine is er een punt aangevijld. Het boutje is in de connector geschroefd en de aluminium grondplaat, met daarop de print, is op de bodem van het kastje gezet en stevig tegen de zijkant gedrukt. De scherpe punt maakt een putje in het plastic. Van binnenuit is met een boortje van 1 mm een gaatje geboord, dat vervolgens van buitenaf met 3 mm is opgeboord. Het "puntboutje" wordt nu in de andere connectorkant geschroefd. Door het gemaakte gat zetten we met een boutje de print "losvast", zorgen dat hij goed recht staat, en oefenen opnieuw druk uit. Ook dit putje wordt eerst van binnenuit geboord en dan van buiten met 3 mm opgeboord. Het aftekenen van de grote opening voor de RS232-connector is met een "bracket" (metalen bevestigingsdeel) van een PC-insteekkaart gedaan. De print zit op drie punten vast, twee boutjes bij de RS232 connector en aan de andere kant is het aluminium plaatje met een boutje vastgezet op de bodem van het kastje.

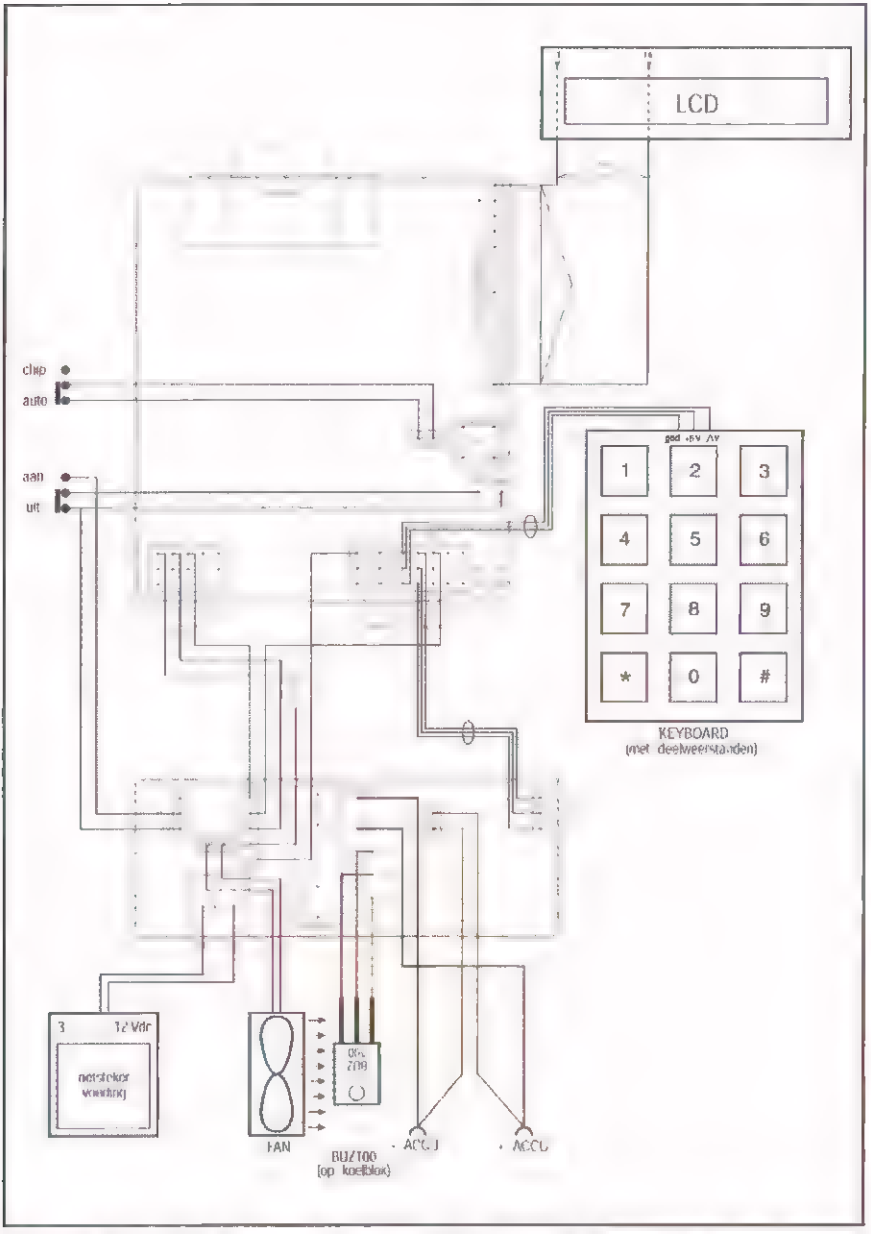
Figuur 9-6: *Een inkijk in de behuizing van de accutester.*

Verbindingen

De NiCad-ontlader is nu bijna klaar, wat nog rest is het maken van enkele verbindingen. Een overzicht van het volledig systeem is getekend in figuur 9-7. Het snoer van de netvoeding komt in K3 (let op de polariteit!). Aan de power-MOSFET solderen we drie draden (neem voor de drain- en source-aansluitingen zwaarder snoer) met aan de andere kant drie schuifbussen, deze komen op de aansluitingen voor T4. Ook van zwaarder snoer maken we de verbindingen tussen de apparaatklemmen met soldeerlippen en +accu en -accu op de ontlaadprint met schuifbussen. Voor alle andere verbindingen worden connectors gebruikt, die worden gezaagd van een female printheader strip. Aan het K6-snoertje komen aan de andere kant twee soldeerlippen die (met de andere soldeerlippen) op de apparaatklemmen worden vastgeschroefd. De ventilator komt op K2 en de stekker van het voedingssnoertje (is een Futaba servosnoertje) van Chip op K1. Voor de verbinding van K5 met **Input 1** van Chip gebruiken we een 3-aderig snoertje met aan beide kanten een 3-polige female header. Via dit snoertje lopen ook de +5 V en 0 V verbindingen. Op K4 komt een 5-polige female header met 5 individuele snoertjes, ieder met aan de andere kant een 1-polige female header. Deze worden verbonden met respectievelijk **Out 2**, **Input 2**, **Out 1**, **Out 0** en **PWM** (servo 1). Zo worden aardlussen voorkomen.
Het laatste snoertje dat nodig is, verbindt K5 op Chip met de schuifschakelaar "Chip-Auto" op het bedieningspaneel.

Bediening van de accutester

De bediening van de ontlader is eenvoudig en consequent. Als het apparaat aanstaat, maar niet ontlaadt, staat **"NC Discharger"** op het LCD. Door op de * te drukken verschijnt keuzemenu 1, drukt men de toets nogmaals in, keuzemenu 2, enz. Er zijn in totaal zes keuzemenu's. De # is de bevestigingstoets, vergelijkbaar met de Entertoets van een PC-keyboard. Als door een of meer keren indrukken van de * het gewenste keuzemenu is bereikt, wordt dit geactiveerd door de # in te drukken. Als dit een menu is, waar gegevens kunnen worden ingevoerd en de # blijft ingedrukt, dan verschijnen de ingestelde gegevens op het LCD. Na loslaten van de # verschijnen vraagtekens en kunnen nieuwe gegevens worden ingevoerd via het decimale invoerveld van het keyboard. De invoer moet worden bevestigd door op de # te drukken.
Tijdens de bediening loopt op de achtergrond een time-out procedure. Bij een bedieningsfout, of als een ingestelde waarde alleen afgelezen wordt,



Figuur 9-7: *De bedrading tussen de diverse groepen van de accu ontlader.*

hoeft verder niets te worden gedaan, automatisch verschijnt na enkele seconden **"NC Discharger"** weer op het LCD.

De menu's

Er zijn zes keuzemenu's, we zullen ze kort behandelen omdat dan in één moeite door de mogelijkheden (en onmogelijkheden) van de ontlader duidelijk worden. De **"Set"**-menu's dienen voor de invoer van gegevens en gedragen zich vrijwel identiek, de overige menus zijn schakelmenu's.

Set loadcurrent?

Als nu de # wordt ingedrukt en vastgehouden, verschijnt de ingestelde ontlaadstroom op het display. Zodra de toets wordt losgelaten verschijnt **"Current: ???? mA"** op het display en kan een nieuwe waarde worden ingevoerd. Als men de waarde niet wil veranderen, dan kan men even wachten of de # indrukken (ter bevestiging) waarna op het display **"! not accepted !"** verschijnt en vrijwel direct naar de keuzemenu invoer wordt teruggekeerd. De minimale stroom is 100 mA, de maximale 2.500 mA. De instelling gaat in stappen van 10 mA. Als de eenhedenwaarde kleiner is dan 5, wordt de stroomwaarde naar beneden afgerond, bij 5 of hoger wordt de stroomwaarde naar boven afgerond.

Set nr of cells?

Hier kan het aantal cellen, waaruit het accupakket bestaat, worden ingevoerd. Dit aantal is minimaal 1 en maximaal 20.

Set volts/cell?

Deze keuze dient om de spanning op te geven, waarbij een cel ontladen is. De waarde mag liggen tussen 0,5 V en 2,5 V. Voor NiCad's die worden ontladen met een stroom kleiner dan 0,5 C is 1,1 V een goede afslagwaarde, bij hogere ontlaadstromen kan 1 V of 0,9 V worden gekozen in verband met de spanningsval ten gevolge van de inwendige weerstand. Door het ruime spanningsbereik kunnen ook loodaccus worden ontladen.

Start discharge?

Na bevestiging met de # wordt getest of de spanning van de accu hoger is dan de afslagwaarde (= aantal cellen x spanning/cel). Als dat het geval is, wordt het ontlaadproces gestart. Eerst wordt het samplinggeheugen (**"Clearing memory!"**) gewist, vervolgens wordt de stroom tot de ingestelde waarde verhoogd (**"Setting Current!"**). Als deze waarde is bereikt,

wordt de ventilator aangezet en de tijd op nul gezet. Tijdens het ontladen worden afwisselend de tijd (**"Time: 00h 00m 00s"**) en de ontladen capaciteit (**"Unload: 0000 mAh"**) op het display getoond, ieder steeds gedurende twee seconden, zodat men de seconden kan zien verspringen. Tijdens het ontlaadproces blijven de keuzemenu's actief, met dien verstande dat men de ingestelde waarden kan zien, maar niet kan veranderen.
Het ontlaadproces gaat door tot de ingestelde spanning is bereikt of ...

Stop discharge?

Hiermee kan het ontlaadproces voortijdig worden beëindigd.

See last result?

Als het ontlaadproces is beëindigd, worden afwisselend de ontlaadtijd en de gemeten capaciteit op het LCD getoond en worden geluidssignalen gegeven. Als echter op een toets wordt gedrukt, verschijnt **"NiCad Discharger"** op het display en verdwijnt het resultaat van de meting. Dit keuzemenu laat het resultaat van de laatste meting weer op het LCD zien.

Software

De NiCad-ontlader kan op twee manieren werken: de commandoprocessor van Chip kan actief zijn of een programma kan draaien. Om met het apparaat te communiceren of om programma's te laden moet Chipterm.exe op de PC draaien en de ontlader met de seriële poort zijn verbonden. De schakelaar "Chip-Auto" op de ontlader wordt op Chip gezet en de ontlader wordt aangezet. Nu kan een programma worden geladen door Alt+I in te drukken. Onderin het scherm wordt naar de naam van het programma gevraagd. Het achtervoegsel is altijd .hex. Na bevestiging met Enter wordt het programma geladen. Door het programma Keytest.asm (listing 3-6) te assembleren en te laden kan het keyboard worden getest. Het programma kan worden gestart met het commando **chip** of door de Chip-Auto schakelaar op **Auto** te zetten en de ontlader aan te zetten.

Verzwakker testen

De verzwakker kan worden getest met het programma Divitest.asm, zie listing 9-1. Op de apparaatklemmen zetten we een spanning van 5 Vdc. De waarde op het display moet 125 bedragen (5/10,2 x 255) en de verzwakking 2. Bij een ingangsspanning van 15 Vdc moet de waarde 188 bedragen en de verzwakking 4.

```
; Listing 9.1: Divitest.asm
; support program for NiCad discharger to test voltage divider
;
; measure NC voltage, set divider if necessary and show divider
; setting and
; analog voltage once a second on the LCD display.
; use output flag f, which is set every second by the operating
; system.
;
ncdivts   p = ncdivt2      ; point p to display text
          ld 0,f           ; load initial display
          p = a-stack      ; point p to a stack
ncdivt1   call getvsub     ; call get voltage sub
          skip out f = 0   ; wait for f to be set
          call showsub     ; f is set, call showsub
          jp ncdivt1       ; and loop
ncdivt2   asciz "Att :2, Val= ???"
;
getvsub   res out 0        ; reset divider 0
          res out 1        ; reset divider 1
          v1 = 32          ; v1 = divider setting 2
          call voltsub     ; get voltage into v0
          skip v0 = ff     ; skip if reached max voltage
          ret              ; not max, return
          set out 0        ; set divider 0
          v1 = 34          ; v1 = divider setting 4
          call voltsub     ; get voltage into v0
          skip v0 = ff     ; skip if reached max voltage
          ret              ; not max, return
          res out 0        ; reset divider 0
          set out 1        ; set divider 1
          v1 = 38          ; v1 = divider setting 8
          call voltsub     ; get voltage into v0
          ret
;
voltsub   v0 = ana 2
          ret
;
showsub   res out f; reset seconds flag
          v2 = 02
          v2 to tone       ; give short beep
          v1,v1 to mp      ; copy divider setting to mp
          ld 5,5           ; show divider setting
          v0 to 3 dec mp   ; convert v0 into 3 dec chars
          ld d,f           ; show 3 dec chars on display
          ret              ; and return
```

***Listing 9-1:** Het programma Divitest.asm waarmee u de verzwakker kunt testen.*

Tot slot 25 Vdc, de waarde moet 156 zijn en de verzwakking 8. De gemeten waarden mogen 2 of 3 eenheden afwijken, maar als dat meer is, en de delerweerstanden zijn juist, dan is waarschijnlijk T2 of T3 lek.

Accutest.asm

Als de tests met succes zijn afgerond, kan het programma Accutest.asm, zie listing 9-2, worden geassembleerd en geladen. Hiermee is de NiCad-ontlader gebruiksklaar op afregeling van de stroom na. Sluit een accupakket op de ontlader aan, met in serie een stroommeter. Stel het aantal cellen in, de spanning per cel en de stroom op 2.500 mA. Regel met P1 de stroom op 2.500 mA. Stel nu de stroom in op 100 mA en regel met P2 de stroom op deze waarde af. Door de regellus fluctueert de aflezing, regel daarom af op een zo goed mogelijk gemiddelde.

```
; Listing 9.2: Accutest.asm
; Chips NiCad discharger and capacity measurer
;
; in 0      = keyboard
; in 1      = unload current
; in 2      = NiCad voltage from divider
; out 0     = set divider :4
; out 1     = set divider :8
; out 2     = cooling fan
; out 4     = toggle for checking with scope
; out 6     = increasing current to set value flag
; out 7     = see last result flag
; out 8     = current motorboat flag
; out 9     = run flag
; out a     = error flag
; out b     = enter selected routine after key up flag
; out c     = wait on flag
; out d     = key down flag
; out e     = select flag
; out f is set every second by the operating system
; v0-v5     = general purpose variables, used by selecting system
; v0 v2     = during run time used by selecting system to show
; set values
; v3 v5     = used for reading hours, minutes, seconds of the
; real time clock
; v6        = mAh counter 0
; v7        = mAh counter 1
; v8        = current/second adding register 0
; v9        = current/second adding register 1
; va        = current set in mA * 10
; vb        = pwm value
; vc        = voltage set
; vd        = seconds sequencer
```

```
; ve       = selector
; vf is carry/borrow/overflow flag + general purpose
;
; artimer (pwm-timer) register adressen, port b data register
; and some other
drb       equ 8c1          ; port b data register
arsc1     equ 8d7          ; ar status control register 1
arrc      equ 8d9          ; ar reload register
arcp      equ 8da          ; ar compare register
armc      equ 8db          ; ar mode control register
clkhrs    equ 8ac          ; real time clock hours register
spoint1   equ 8bb          ; saved pointer high byte
;
          p = initvar
          vd,ve = mp       ; initialise vd...ve
main 0    set out 4        ; set out 4 for checking with scoop
          skip out 9 = 0   ; skip if discharger not running
          call cadjust     ; call current adjust
          res out 4        ; reset out 4 for checking with scoop
main 1    skip out d = 0   ; skip if .not. key down
          jp keydown       ; key beeing pressed, jump to keydown
          skip out e = 0   ; skip if .not. select active
          jp select1       ; select active, jump to select
          skip ve <> ff    ; skip if ve .not. active (= ff)
          jp main_3        ; select .not. active, jump to
                           ; continu main loop
          jp getkeyn       ; select is active, try to get a
                           ; key value
main 2    skip ve <> 00    ; jump according to selector value ve
          jp setcur1
          skip ve <> 10
          jp setnum1
          skip ve <> 20
          jp setvol1
main 3    v0 = key 0       ; get a key value
          skip v0 <> 3a    ; skip if .not. 3a (= *)
          jp select0       ; got a key value 3a, jump to select0
          skip out f = 1   ; skip if on the seconds flag
          jp main_0        ; no seconds flag, loop to main_0
;
; distribution in time (every second)
;
seconds   res out f        ; reset seconds flag
          skip out 9 = 1   ; skip if discharger is running
          jp intrmsg       ; not running, jump to intro message
          skip out 6 = 0   ; skip if current is increasing
                           ; towards  set value
          jp inccurr       ; jump to check if current has
                           ; increased
          skip out 8 = 0   ; skip if current is stable
          jp boating       ; current is motor boating, jump
          v0 = ana 2       ; v0 is Nicad voltage
```

```
            vf = sectimer     ; get seconds timer
            skip vf <> 00     ; skip if still running
            call logtime      ; zero, time to take log sample
            v0 - vc           ; if v0 < vc (= voltage set) then
                              ; vf = 01
            skip vf = 00      ; skip if vNiCad >= vset
            jp ncready        ; equal or smaller, jump
            call addcurr      ; add mAs to current adding registers
            skip vf <> 00     ; skip if 1 mAh is reached
            jp second0        ; not yet, jump to continu
            v6 + 01           ; add 1 mAh to mAh counter 0
            skip v6 = 64      ; skip if not yet reached 100d
            jp second0
            v6 = 00           ; load mAh counter 0 with 00
            v7 + 01           ; mAh counter 1 + 1
second0     vd + 01           ; increment seconds sequencer
            skip vd = 01
            jp second1
            call elapsed      ; load template and show elapsed
                              ; time on display
            jp main_0
second1     skip vd = 02
            jp second2
            call elapse1      ; show elapsed time, without loading
                              ; template
            jp main_0
second2     skip vd = 03
            jp second3
            call showmah      ; load template and show unloaded mAh
            jp main_0
second3     skip vd = 04
            jp second4
            call showma1      ; show unloaded mAh, without loading
                              ; template
            skip out 7 = 1
            vd = 00
            jp main_0
second4     vf = 14           ; give a short beep
            vf to tone
            p = slrttxt       ; point to see last result display
            ld 0,f            ; load display
            vd = 00           ; set sequencer to 00
            jp main_0
;
intrmsg     skip out a = 0    ; skip if error flag is not set
            jp intrms1        ; error flag is set, jump
            skip out 7 = 0    ; skip see last result flag not set
            jp second0        ; show last result
            p = titltxt       ; display title and go back
            ld 0,f
            jp main_0
intrms1     v0 - timer        ; get timer
```

```
          skip v0 <> 00     ; skip if .not. 00
          res out a         ; time out, reset error flag
          jp main_0         ; and go back
;
inccurr   v0 = ana 1        ; v0 = current voltage
          v1 = 06           ; lower limit of current voltage
          skip v0 > v1      ; skip if current voltage > lower
                            ; limit
          jp main_0
          res out 6         ; reset setting current flag
          set out 2         ; start cooling fan
          p = nulline       ; point to line with "00" bytes
          v3,v9 = mp        ; load into v3...v9 (v6...v9 = mAh
                            ; registers)
          p = clkhrs        ; point to real time clock hours
                            ; register
          v3,v5 to mp       ; and set all clock registers to zero
          v3 to sectimer    ; set sectimer to 00
          jp main_0
;
boating   p = curmtxt
          ld 0,f
          v3 = 04
          v3 to tone
          jp main_0
;
ncready   v0 + vc           ; v0 is Nicad voltage - vc, add vc to
                            ; restore voltage
          call logtime      ; write it at the end of logging
                            ; memory space
          p = saveres       ; point to save result storage
          v3,v7 to mp       ; save result in storage (= elapsed
                            ; time and mAh)
          son               ; start ar-timer in servo mode..
          soff              ; ..and stop ar timer
          res out 2         ; stop cooling fan
          res out 9         ; reset run flag
          res out 6         ; in case battery got flat while
                            ; current was settling
          set out 7         ; set see last result flag
          vf = 80
          vf to tone        ; give beep
          jp main_0
;
; keydown, jump routine to wait for key release or time out
;
keydown   v0 = key 0        ; v0 is key value
          skip v0 = 3f      ; skip if no key pressed
          jp keydwn2        ; jump, key still beeing pressed
          res out d         ; key has been released, reset key
                            ; down
          skip out 9 = 1    ; skip if running
```

```
            jp keydwn1          ; not running, jump to continu
            skip out b = 0      ; running, if out b is 1, we must
                                ; quit
            jp keydwn3          ; jump to quit
keydwn1     skip out b - 0
            jp main_2           ; if out b - 1, enter selected
                                ; routine after key up
            jp main_0           ; else get a key value in via main_0
keydwn2     v0 = timer          ; get time out value
            skip v0 = 00
            jp main_0           ; time still running, jump to main_0
            v0 - 02             ; give a sound signal
            v0 to tone          ; while key beeing pressed
keydwn3     res out b           ; reset select key up flag
            res out c           ; reset wait on flag
            res out e           ; reset select flag
            ve - ff             ; set selector ve to off
            jp main_0           ; and loop to main_0
;
; getkeyn, jump routine to get a numerical key value for passing
; to selected
; jump routine main_2 according to selector ve
;
getkeyn     v0 = key 0          ; get a key value into v0
            skip v0 = 3f        ; skip if no key pressed
            jp getkey2          ; jump, got value
getkey1     v0 = timer          ; get time out value
            skip v0 = 00        ; skip on time out
            jp main_0           ; not yet time out, jump to try to
                                ; get good value
            ve = ff             ; set selector ve to off
            jp main_0           ; and continue main task
getkey2     skip v0 <> 3a       ; skip if key .not. "*"
            jp getkey1
            set out d           ; got key value, set key down
            v1   02
            v1 to tone          ; give key beep
            v1 - 8c             ; circa 5 seconds
            v1 to timer         ; set time out value
            jp main_2           ; jump to selected set routine in
                                ; main_2
;
; select, jump routine to make a choice out of rotating menus
;
select0     v0 = 02
            v0 to tone          ; give a key beep
            p = sel?txt         ; display the ?
            ld f,f              ; at position f
            p = selctxt         ; set p to the first of the
                                ; selections
            ld 0,e              ; load chars 0-e
            ve - 00             ; preset the selector to 00
```

```
          set out e          ; set selection to active
          set out d          ; set key down to active
          res out 7          ; reset see last result flag
          v0 = 5c
          v0 to timer        ; load time out value
          jp main_0          ; jump back to main_0
select1   skip out c = 0     ; entry point for rotate selections
          jp select2         ; jump if time out value already
                             ; loaded
          v0 = 5c            ; out c = 0, load time out value
          v0 to timer
          set out c          ; and set out c
select2   v0 = key 0         ; get a key value
          skip v0 <> 3a
          jp select3         ; jump to select3 if key
                             ; value = 3a (*)
          skip v0 <> 3b
          jp select4         ; jump to select4 if key
                             ; value = 3b (#)
          v0 = timer         ; get time out value
          skip v0 = 00       ; skip if time is out
          jp main_0          ; not yet, keep trying to get
                             ; 3a or 3b
          res out e          ; time is out, reset selector
                             ; (no more active)
          res out c          ; reset wait on
          jp main_0          ; and loop to perform main tasks
select3   v0 = 02            ; got key value 3a (*)
          v0 to tone         ; give key beep
          skip ve <> 50      ; skip if .not. last selector value
          ve = f0            ; yes, last, preset to f0
          ve + 10            ; add 10 to selector value
                             ; (last will become 00)
          p = selctxt        ; set pointer to first selection text
          p + ve             ; add selector offset to p
          ld 0,e             ; and put text on display
          res out c          ; reset wait on flag
          set out d          ; set keydown flag
          v0 = 5c
          v0 to timer        ; load time out value
          jp main_0          ; loop to cycle through selection
                             ; while .not. time out
select4   v0 = 02            ; got a key value 3b (#)
          v0 to tone         ; give key beep
          res out c          ; reset wait on
          res out e          ; reset selector flag (disable jump
                             ; to select)
          set out d          ; set key down flag
          v0 = 8c            ; 5 seconds
          v0 to timer        ; set time-out value
          skip ve <> 00      ; and jump to entry point of set
                             ; routines
```

```
          jp setcurr        ; according to selector value ve
          skip ve <> 10
          jp setnumb
          skip ve <> 20
          jp setvolt
          skip ve <> 30
          jp setstrt
          skip ve <> 40
          jp setstop
          skip ve <> 50
          jp setseel
          break             ; this point must never be reached
;
; setcurr, jump routine to set current
;
setcurr   p = currtxt       ; entry point from select routine
          ld 0,f            ; point p to text, put on display
          p - savecur       ; point to saved current byte
          v2,v2 = mp        ; get byte into v2
          p = a-stack       ; load display
          v2 to 3 dec mp    ; convert current to 3 decimals
          ld 9,b            ; load display
          set out b         ; set out b for entry after key
                            ; release
          jp main_0         ; jump to main_0 to perform other
                            ; tasks
setcur1   skip out b - 1    ; entry point from keydown and
                            ; main routine
          jp setcur2        ; if not entry after key release
                            ; then jump
          res out b         ; entry after key release, reset flag
          p = queline       ; point to question marks
          ld 9,c            ; load display
          p = zerline       ; point to ascii zero's (30)
          v2,v5 - mp        ; v2...v5 - 30
          p = inpline       ; point to input line
          v2,v5 to mp       ; clear input line (make them all
                            ; zero's)
          jp main_0         ; jump to main_0 to perform other
                            ; tasks
setcur2   skip v0 <> 3b     ; skip if key <> "#"
          jp setcur3        ; jump to check received key values
          v3 to v2          ; shift received key strokes
          v4 to v3
          v5 to v4
          v0 to v5
          v2,v5 to mp       ; copy v2...v5 into input line
          ld 9,c            ; show on display
          jp main_0         ; jump to main_0 to perform other
                            ; tasks
setcur3 vd = 00             ; reset seconds sequencer
          ve = ff           ; set selector ve to off
```

```
          p = a-stack       ; point to a-stack
          v2,v4 to mp       ; and save received keystrokes
                            ; except the last
          call chek250      ; check if value < 250d
          skip vf = 00
          jp notaccp        ; vf = 01, value > 250, jump to not
                            ; accepted
          v0 = 3dec mp      ; convert to byte, use v0
          v0 + f6           ; add f6 to check if v0 >= 0a
          skip vf <> 00     ; if vf <> 00, then v0 >= 0a
          jp notaccp        ; jump to not accepted
          v0 + 0a           ; restore v0
          v5 + cb           ; check if last number >= 5
                            ; (ascii 35)
          skip vf = 00      ; skip if not
          v0 + 01           ; yes, increment current by 01
          p = savecur       ; point to storing place for current
                            ; byte
          v0,v0 to mp       ; save current
          jp main_0         ; jump to main task
;
; setnumb, jump routine to set number of cells
;
setnumb   p = numbtxt       ; entry point from select routine
          ld 0,f            ; point p to text, put on display
          p = savenum       ; point p to current byte
          v2,v2 = mp        ; v2 = current byte
          p = a stack       ; load display
          v2 to 3 dec mp    ; convert current to 3 decimals
          p + 1             ; point to tens
          ld e,f            ; load display
          set out b         ; set out b for entry after key
                            ; release
          jp main_0         ; jump to main_0 to perform other
                            ;  tasks
setnum1   skip out b = 1    ; entry point from keydown and main
                            ; routine
          jp setnum2        ; if not entry after key release then
                            ; jump
          res out b         ; entry after key release, reset flag
          p = queline       ; point to question marks
          ld e,f            ; load display
          p = zerline       ; point to zero's (30)
          v2,v5 = mp        ; v2...v5 = 30
          p = inpline       ; point to input line
          v2,v5 to mp       ; clear input line (all zero's)
          p + 1
          jp main_0         ; jump to main_0 to perform other
                            ; tasks
setnum2   skip v0 <> 3b     ; skip if key <> "#"
          jp setnum3        ; jump to check received key values
          v4 to v3          ; shift received key strokes
```

```
          v0 to v4
          v3,v4 to mp       ; copy v3...v4 into input line
          ld e,f            ; show on display
          jp main_0         ; jump to main_0 to perform other
                            ; tasks
setnum3   vd = 00           ; reset seconds sequencer
          ve   ff           ; set selector ve to off
          p = inpline       ; point to start of inputline
          v2,v4 = mp        ; get 3 numbers (first = 30)
          p = a-stack       ; point to a-stack
          v2,v4 to mp       ; put them on a-stack
          v0 = 3dec mp      ; convert to byte, use v0
          skip v0 <> 00     ; minimum cell number = 1
          jp notaccp
          v0 + eb           ; add eb to check if v0 > 14 (20d)
          skip vf = 00      ; if vf <> 00, then v0 > 14
          jp notaccp
          v0 + 15           ; restore v0
          p = savenum       ; point to savenum
          v0,v0 to mp       ; copy value into savenum
          jp main_0         ; jump to main task
;
; setvolt, jump routine to set cell voltage
;
setvolt   p = volttxt       ; entry point from select routine
          ld 0,f            ; point p to text, put on display
          p = savevol       ; point to saved voltage byte
          v2,v2 = mp        ; get byte into v2
          p = a-stack       ; load display
          v2 to 3 dec mp    ; convert voltage to 3 decimals
          ld b,b            ; load units
          p + 1
          ld d,d            ; load tenths
          set out b         ; set out b for entry after key
                            ; release
          jp main_0         ; jump to main_0 to perform other
                            ; tasks
setvol1   skip out b = 1    ; entry point from keydown and main
                            ; routine
          jp setvol2        ; if not entry after key release then
                            ; jump
          res out b         ; entry after key release, reset flag
          p = queline       ; point to question marks
          ld b,b
          ld d,d            ; load display
          p = zerline       ; point to zero's (30)
          v2,v5 = mp        ; v2...v5 = 30
          p = inpline       ; point to input line
          v2,v5 to mp       ; clear input line (all zero's)
          jp main_0         ; jump to main_0 to perform other
                            ; tasks
setvol2   skip v0 <> 3b     ; skip if key <> "#"
```

```
          jp setvol3        ; jump to check received key values
          v3 to v2          ; shift received key strokes
          v0 to v3
          v2,v4 to mp       ; copy v2...v4 into input line
          ld b,b            ; put units on display
          p + 1             ; point to tenths
          ld d,d            ; load tenths
          p = inpline       ; point to begin of input line
          jp main_0         ; jump to main_0 to perform other
                            ; tasks
setvol3   vd   00           ; reset seconds sequencer
          ve   ff           ; set selector ve to off
          p = a-stack       ; point to a-stack
          v2,v4 to mp       ; and save received keystrokes
          call chek250      ; check if value < 250d
          skip vf = 00
          jp notaccp        ; vf = 01, value > 250d
          v0 = 3dec mp      ; convert to byte, use v0
          v0 + ce           ; add cd to check if v0 >= 32h
                            ; (0,50 V)
          skip vf <> 00     ; if vf = 01, then v0 > 32
          jp notaccp
          v0 + 32           ; restore v0
          p = savevol       ; point to storing place for voltage
          v0,v0 to mp       ; save voltage
          jp main_0         ; jump to main task
;
; setstrt, jump routine to start discharger
;
setstrt   vd - 00           ; reset seconds sequencer
          ve = ff           ; set selector to off
          skip out 9 = 0    ; skip if .not. running
          jp main_0         ; running, jump back
          call vcalcul      ; calculate discharge voltage and
                            ; set dividers
          v0 = ana 2        ; v0 = NiCad voltage
          skip v0 > vc      ; skip if V Nicad > voltage set
          jp flatpac        ; jump to display empty or not
                            ; connected
          p = savecur       ; point to saved current
          va,va = mp        ; load into va
          call clrmemo      ; call clear log memory space
          p = logstrt       ; point to start of logging memory
          save p            ; ...and save pointer
          vb = 01           ; set pwm value to 01
          call arstart      ; start pwm-timer
          res out 8         ; reset motor boating current flag
          set out 9         ; set run flag
          set out 6         ; set increase current flag
          p = incctxt       ; point to setting current text
          ld 0,f            ; show text on lcd
          jp main_0
```

```
;
; setstop, jump routine to stop start discharger
;
setstop   skip out 9 - 1   ; skip if discharger is running
          jp notrunn       ; if not, display not running text
          son              ; use son and soff to stop pwm-timer
          soff
          vd = 00          ; reset seconds sequencer
          ve = ff          ; set selector to off
          res out 6        ; in case battery went suddenly flat
          res out 8        ; reset motor boating current flag
          res out 9        ; reset run flag
          res out 2        ; stop cooling fan
          jp uhalted       ; jump to display message and beep
;
; setseel, jump routine to see last result on display when not
; running
;
setseel   vd = 00          ; reset seconds sequencer
          ve = ff          ; reset selector
          skip out 9 = 0   ; skip if .not. running
          jp main_0        ; running, do nothing, jump back
          p = saveres      ; point to saved result
          v3,v7 = mp       ; load into v3...v7
          set out 7        ; set see last result flag
          jp main_0
;
notrunn   p = notrtxt      ; point to not running text
          skip a           ; (skip a = skip always)
notaccp   p = notatxt      ; point to not accepted text
          skip a
flatpac   p = flattxt      ; point to flat pack text
          skip a
uhalted   p = halttxt      ; point to halted by user text
          set out a        ; set error flag
          ld 0,f           ; show error text
          v0 = 1c          ; beeptime
          v0 to tone
          v1 = 1c          ; display error message time
          v1 to timer
          jp main_0
;
; chek250, subroutine to check if v2 (hundreds), v3 (tens),
; v4 (units) > 250
; if yes, vf = 01. Remember these are ascii values
;
chek250   v2 + cd          ; hundreds + cd
          skip vf = 00     ; skip if hundreds <= 32
          ret              ; hundreds > 32, return vf = 01
          skip v2 = ff     ; skip if hundreds was 32
          ret              ; hundreds < 32, return vf = 00
          v3 + ca          ; tens + ca
```

```
          skip vf = 00      ; skip if tens <= 35
          ret               ; tens > 35, return vf   01
          skip v3 - ff      ; skip if tens was 35
          ret               ; tens < 35, return vf   00
          v4 + cf           ; units + cf
          ret               ; if units > 0 then vf   01
;
; arstart, subroutine to start artimer in pwm mode
;
arstart   p = pwminit       ; point to pwm initialize values
          v3,v8 = mp        ; load into v3...v8
          p - armc          ; point to ar mode control register
          v3,v3 to mp       ; stop artimer, it could be running
          p = drb           ; point to port b dataregister
          v3,v3 = mp        ; v3 = port b data register
          v4 or v3          ; we don't want to change the
                            ; outputs, so we .or.
          v4,v4 to mp       ; select servo 1 (11?? ????b)
          p = arsc1         ; point to artimer status control
                            ; register 1
          v5,v5 to mp       ; load predivider ratio (00h - :1)
          p = arrc          ; point to ar reload register
          v6,v6 to mp       ; load reload value (00h)
          p = arcp          ; point to ar compare register
          v7,v7 to mp       ; load ar compare value (ffh)
          p = armc          ; point to ar mode control register
          v8,v8 to mp       ; start pwm generation
          ret
pwminit   bytes 00c00000ffe0
;
; cadjust, check discharge current and adjust if necessary
;
cadjust   v0 = ana 1        ; get current voltage
          skip v0 <> va     ; skip if current measured <> cset
          ret
          skip v0 > va      ; skip if measured > cset
          jp cadjus1        ; jp if measured < cset
          vb + ff           ; decrease pwm-value
          skip a
cadjus1   vb + 01           ; increase pwm-value
          skip vb <> 00     ; skip if vb .not. 00
          set out 8         ; vb = 00, current is motorboating,
                            ; set error flag 8
          v0 = ff           ; v0 = xor value 1
          v0 xor vb         ; vb = pwm value, invert by .xor.
          p = arcp          ; point to ar compare register and..
          v0,v0 to mp       ; load v0 into ar compare
          ret
;
; vcalcul, calculate voltage end value from number of cells and
; voltage/cell
; if necessary set appropiate divider. If no divider or divider
```

```
; 1 is set and
; the culculated value > a0 then value = value/2, next higher
; divider is set.
; load value into vc (voltage set)
;
vcalcul   res out 0        ; reset divider :4
          res out 1        ; reset divider :8
          p = savenum      ; point to saved number of cells
          v3,v3 = mp       ; load into v3
          p = savevol      ; point to saved voltage/cell
          v4,v4 = mp       ; load into v4
          p = a-stack
          v3 * v4 to mp    ; multiply number of cells by
                           ; voltage/cell
          v5 = 04          ; v5 = dividing factor 1
          vc = mp/v5
          skip vf <> 00    ; skip on overflow
          jp vcalcu1       ; jump, result fits
          v5 = 08          ; v5 = dividing factor 2
          vc = mp/v5
          skip vf <> 00    ; skip on overflow
          jp vcalcu2       ; jump, result fits
          v5 = 10          ; v5 = dividing factor 3
          vc = mp/v5
          set out 1        ; set divider to :8
          v9 = 08          ; divider setting = 08
          ret
vcalcu1   v9 = 02          ; divider setting = 02
          v3 = a0          ; result 1 fits, test if > a0
          skip vc > v3
          ret
          v3 = 01          ; result 1 > a0
          vc * v3 to mp    ; multiply vc by 01, to put it
                           ; on mp
          v3 = 02
          vc = mp/v3       ; divide value by 02
          set out 0        ; set divider to :4
          v9 = 04          ; divider setting = 04
          ret
vcalcu2   set out 0        ; set divider to :4
          v9 = 04          ; divider setting = 04
          v3 = a0          ; result 2 fits, test if > a0
          skip vc > v3
          ret
          v3 = 01          ; result 2 > a0
          vc * v3 to mp    ; multiply vc by 01, to put it
                           ; on mp
          v3 = 02
          vc = mp/v3       ; divide value by 02
          res out 0        ; res divider :4
          set out 1        ; set divider to :8
          v9 = 08          ; divider setting = 08
```

```
          ret
;
; elapsed, show elapsed discharge time on display
;
elapsed   p = elaptxt       ; point to initial elapsed time
                            ; display
          ld 0,f            ; put on display
elapse1   p = clkhrs        ; point to clock hours
          skip out 7 = 1    ; skip if see last result flag = 1
          v3,v5 = mp        ; load hours, minutes, seconds into
                            ; v3...v5
          p = a stack       ; point to a stack
          v3 to 3dec mp     ; convert v3, v4 and v5 into decimals
          p + 1             ; and show them on display
          ld 5,6
          p = a-stack
          v4 to 3dec mp
          p + 1
          ld 9,a
          p = a-stack
          v5 to 3 dec mp
          p + 1
          ld d,e
          ret
;
; showmah, show unloaded mAh
;
showmah   p = smahtxt
          ld 0,f
showma1   p = a-stack
          v7 to 3dec mp
          p + 1
          ld 8,9
          p = a-stack
          v6 to 3dec mp
          P + 1
          ld a,b
          ret
;
; subroutine addcurr
; first: adding regs + current set
; then:  adding regs + fe98h
; if there was a carry, we passed 168h (360d) border, the new
; value of the
; adding regs - adding regs - 168h, wich is right and vf - 01
; (<>00)
; if there was no carry, we adjust the adding regs by adding
; 0168h and
; make vf = 00
;
addcurr   v8 + va           ; add curset to adding reg 0
          skip vf = 00      ; skip if no carry
```

```
            v9 + 01             ; carry, adjust adding reg 1
            v8 + 98             ; add 98 to adding reg 0
            skip vf = 00        ; skip if no carry
            v9 + 01             ; carry, adjust adding reg 1
            v9 + fe             ; add fe to adding reg 1
            skip vf   00        ; skip if no carry to adjust adding
                                ; regs
            ret                 ; overflow is new value, vf is 01,
                                ; return
            v8 + 68             ; adjust adding reg 0 by adding 68
            skip vf - 00        ; skip if no carry
            v9 + 01             ; carry, adjust adding reg 1
            v9 + 01             ; adjust adding reg 1 by adding 01
            vf   00             ; be sure to make vf 00 for not yet
                                ; 1 mAh
            ret
;
; clrmemo, fill logging space with divider setting, current set
; and "00" bytes
;
clrmemo     p = clrmtxt         ; point to clear memory text
            ld 0,f              ; load dislay
            p - diviset         ; point to divider storage
            v9,va to mp         ; save divider and current settings
            v0 = 00             ; v0 - filling byte
            p = logstrt         ; point to start of log memory
clrmem1     save p              ; save pointer
            p = spoint1         ; point to saved pointer high byte
            vf,vf = mp          ; vf = saved pointer high byte
            skip vf <> 08       ; skip if not reached end
            ret
            rest p              ; restore saved pointer
            v0,v0 to mp         ; write filling byte
            p + 1               ; increment pointer
            jp clrmem1          ; loop until ready
;
; logtime, reload sectimer, check memory space, if space then
; write v0
;
logtime     vf = 3c             ; 1 minute
            vf to sectimer      ; reload seconds timer
            p = spoint1         ; point to saved pointer high byte
            vf,vf = mp          ; vf = saved pointer high byte
            skip vf <> 08       ; skip if not reached end
            ret
            rest p              ; restore saved pointer
            v0,v0 to mp         ; write voltage byte
            p + 1               ; increment pointer
            save p              ; ... and save pointer
            ret
;
inpline     bytes 00000000
```

```
zerline   asciz "0000"
queline asciz "????"
nulline bytes 00000000000000
;
initvar   bytes 00ff       ; vd = 00, ve = ff
savecur   bytes 6400       ; initial current = 1000 mA
savenu    bytes 0400       ; initial number of cells = 4
savevol   bytes 6400       ; initial volts/cell = 1,00
saveres   bytes 0000000000 ; saved result from v3...v7
;
titltxt   asciz "NiCad Discharger"   ; main title
selctxt   asciz "Set loadcurrent"    ; selector ve   00
          asciz "Set nr of cells"    ; selector ve = 10
          asciz "Set  volts/cell"    ; selector ve = 20
          asciz "Start discharge"    ; selector ve = 20
          asciz "Stop  discharge"    ; selector ve = 40
          asciz "See last result"    ; selector ve = 50
sel?txt   asciz "?"                  ; last of selection text
currtxt   asciz "Current: ???0 mA"   ; current display
numbtxt   asciz "Number cells: ??"   ; number of cells display
volttxt   asciz "Volt/cell: ?,? V"   ; volts/cell display
notatxt   asciz "! not accepted !"   ; not accepted display
flattxt   asciz "! open or flat !"   ; open or flat pack display
elaptxt   asciz "Time   h   m   s"   ; elapsed time display
curmtxt   asciz "Current unstable"   ; motorboat current display
smahtxt   asciz "Unload:      mAh"   ; unloaded mAH display
slrttxt   asciz "Ready see result"   ; see last result display
halttxt   asciz "!halted by user!"   ; halted by user display
notrtxt   asciz "! *** idle *** !"   ; idle display
clrmtxt   asciz "Clearing memory!"   ; clear log memory display
incctxt   asciz "Setting current!"   ; increasing current disp.
;
          org 600
diviset   bytes 0000   ; byte 1 = divider, byte 2 = current*10
logstrt   bytes 00     ; start of logging space, end = 7ff
```

Listing 9-2: *De listing van Accutest.asm.*

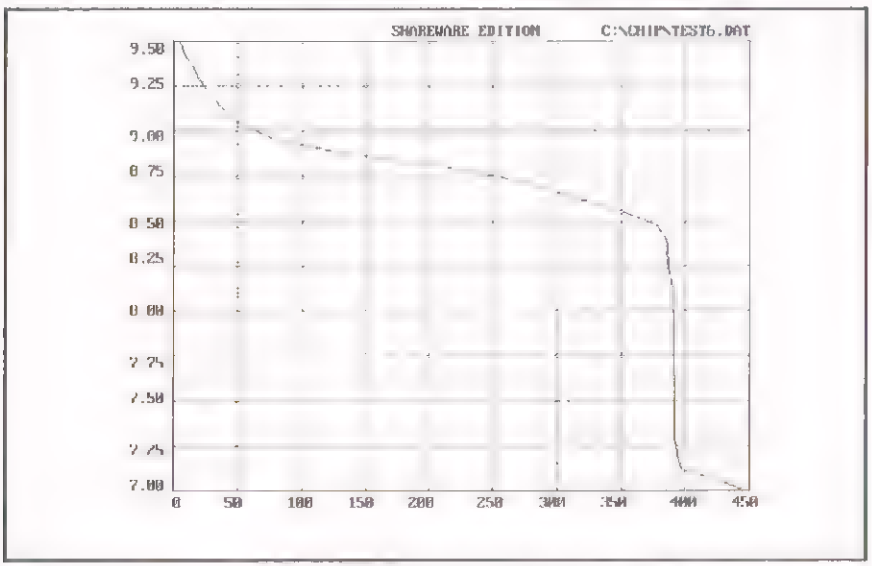
Maken van ontlaadspanningscurves

Tijdens de ontlading wordt iedere minuut een spanningsmonster in het geheugen opgeslagen. Daarvoor wordt het EEPROM-geheugen van 600h-7FFh gebruikt. Op **adres 600h** staat de verzwakkerinstelling en op **601h** de stroomwaarde. Na het ontladen kunnen deze monsters in een bestand worden opgeslagen. Ga daarvoor naar de commandoprocessor en open met Alt+o een .log bestand. Door het commando **prog 600** te geven verschijnt adres 600h op het scherm gevolgd door twee inhoudsbytes. Door de + in te drukken verschijnen volgende adressen. Ga door tot de inhoudsbyte 00h verschijnt. Dat is het einde van de monsters en het

.log bestand kan worden gesloten met Alt+c. Met het programma **Log-dat.exe** (staat ook op de Internetpagina van Chip) worden de monsters omgerekend naar spanningswaarden en genummerd. Bovendien worden de waarden geïnterpoleerd om de curve vloeiender te maken. Tevens worden de gemiddelde spanning en de ontladen energie berekend. Het opgewekte .dat bestand kan in het plotterprogramma worden ingelezen en als ontlaadcurve worden getekend. Het bestand **Plotter.zip** op www.vego.nl/chip bevat alle noodzakelijke gegevens.

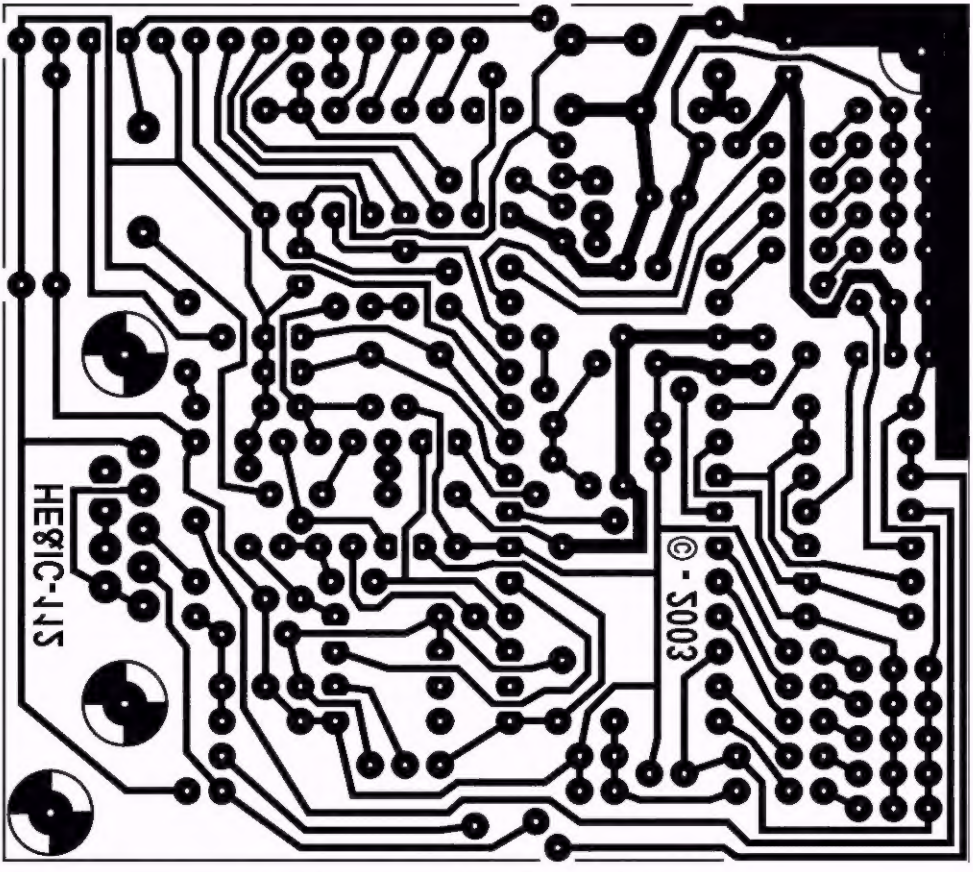
Analyse van een ontlaadspanningscurve

De curve in figuur 9-8 is het ontlaadspanningsverloop van een 7-cellig NiCad-pakket met een capaciteit van 1.900 mAh. De ontlaadstroom was ingesteld op 190 mAh (0,1 C) en de ontlaadspanning op 1 V/Cel. Na 8 uur en 4 minuten was de accu ontladen en de gemeten capaciteit bedroeg 1.532 mAh. Hoewel de accu niet nieuw was, was dat toch minder dan we hadden verwacht. De curve loopt zoals verwacht, maar toch is er iets vreemds. Bij een spanning van 8,4 V is de accu leeg, de spanning valt steil naar beneden, maar bij 7,1 V wordt het spanningsverloop weer vlakker. De spanningsval is ongeveer 1,3 V, blijkbaar is één cel omgepoold en deze cel neemt nu laadstroom op, in omgekeerde richting.

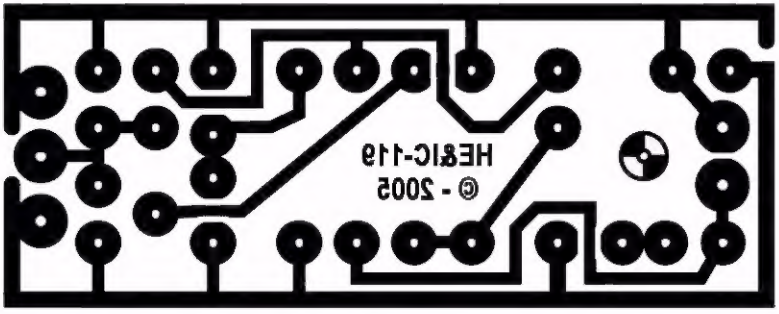


Figuur 9-8: *Het ontlaadspanningsverloop van een 7-cellig NiCad-pakket met een capaciteit van 1.900 mAh.*

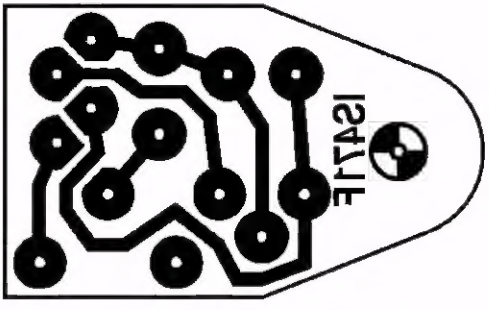
Kort daarna is de eindspanning bereikt. Omdat de ontlaadstroom 0,1 C was, hadden we er beter aan gedaan om de eindspanning op 1,1 V/cel in te stellen. Dan zou er geen ompoling zijn geweest. Loodaccu's zijn gevoeliger voor ompoling dan NiCad's of NiMH-cellen. Gebruikelijk is om voor de eindspanning 75 % tot 80 % van de nominale spanning te kiezen, maar dit hangt natuurlijk ook af van de ontlaadstroom.



afmetingen print: 7,0 cm x 6,2 cm



afmetingen print: 3,0 cm x 1,2 cm



afmetingen print: 1,8 cm x 1,1 cm

Bob Stuurman

Chip

een zelfbouw computertje

156 pagina's
41 illustraties
21 listings

ISBN 90-805610-9-6

NUR 958

SISO 523.1

Chip is een computertje ongeveer zo groot als twee luciferdoosjes tegen elkaar. Het hart van Chip is een microcontroller. Heel bijzonder is, dat in de microcontroller geen programma zit voor een vaste taak, maar een programma dat instructies in een hogere programmeertaal decodeert en ze in microcontroller code uitvoert. Deze programmeertaal is speciaal geschreven voor Chip en heel eenvoudig van opzet. Om met Chip te werken is kennis van microcontrollers niet nodig. Het enige dat nodig is, is Chip's instructieset.

Chip bevat standaard alle hardware die voor eenvoudige toepassingen nodig is en voor de sturing daarvan bevat de hogere programmeertaal instructies.
Om Chip te gebruiken is alleen een PC nodig met een seriële poort. Chip bevat alle software die voor de communicatie nodig is.

Ondanks alle eenvoud is Chip een vrij krachtig computersysteem. Maar om alle mogelijkheden vol te benutten is enige kennis van elementaire digitale principes onmisbaar. Ook begrippen als bytes, nibbles en bits moeten vertrouwd klinken en de elementaire werking van het computermodel volgens von Neumann moet bekend zijn.

Als aan deze voorwaarden wordt voldaan, zal men van Chip veel plezier kunnen hebben en het kleine apparaatje steeds meer gaan waarderen.

ISBN 90-805610-9-6
9 789080 561090

Vego